Informatique musicale

François Pachet, Jean-Pierre Briot

Informatique musicale : du signal au signe musical

Traité IC2, série Informatique et systèmes d'information

Sous la direction de François Pachet et Jean-Pierre Briot

Hermes/Lavoisier

Mai 2004

ISBN: 2-7462-0825-3

Le livre étant épuisé, ceci est une version PDF reconstituée (en 2025)

Table des matières

Introduction 17

Chapitre 1. Méthodes d'analyse du signal musical 21
Boris DOVAL

1.1. Introduction 21
1.2. Outils d'analyse du signal sonore 23
1.2.1. Le signal musical et sa numérisation 23
1.2.2. Analyse à court terme dans le domaine temporel 24
1.2.3. Analyse à court terme dans le domaine fréquenciel 25
1.2.4. Robustesse 28
1.3. Analyse de l'enveloppe temporelle 29
1.3.1. Enveloppe temporelle 29
1.3.2. Estimation de l'enveloppe temporelle 30
1.3.3. Segmentation 32
1.4. Analyse des partiels et du bruit 33
1.4.1. Partiels d'un signal 33
1.4.2. Détection et sélection de pics 34
1.4.3. Interpolation spectrale des pics 36
1.4.4. Suivi de partiels 37
1.4.5. Estimation de la partie bruit 39
1.5. Analyse du fondamental 41
1.5.1. Fréquence fondamentale 42
1.5.2. Concevoir un algorithme de détermination du fondamental 43
1.5.3. Prétraitements 44
1.5.4. Méthodes temporelles 45
1.5.5. Méthodes temporelles à court terme 47
1.5.6. Méthodes fréquencielles à court terme 48
1.5.7. Suivi de la fréquence fondamentale 51
1.6. Analyse de l'enveloppe spectrale 54
1.6.1. Enveloppe spectrale 55

1.6.2. Estimation par prédiction linéaire (LPC) 55
1.6.3. Estimation par cepstre discret 57
1.6.4. Formants 58
1.7. Suivi de notes et applications 58
1.7.1. Quantification de la fréquence fondamentale 60
1.7.2. Détection de début et de fin de notes 60
1.7.3. Suivi de partition 61
1.8. Bibliographie 62

Chapitre 2. La synthèse additive 67
Sylvain MARCHAND

2.1. Le modèle additif 68
2.1.1. Représentation des sons 69
2.1.2. Transformations sonores 74
2.1.3. Modèles dérivés 77
2.2. Synthèse des partiels en temps réel 77
2.2.1. Oscillateurs logiciels 78
2.2.2. Utilisation de la transformée de Fourier inverse 82
2.2.3. Résultats comparatifs 89
2.3. Contrôle des partiels 92
2.3.1. Variation des paramètres 92
2.3.2. Rééchantillonnage par les splines 97
2.4. Accélération par la psycho-acoustique 99
2.4.1. Echelles perceptives 100
2.4.2. Seuil d'audibilité 101
2.4.3. Phénomène de masquage 101
2.4.4. Algorithme général 103
2.5. Techniques non linéaires 105
2.5.1. Remarque préliminaire 106
2.5.2. Synthèse par modulation d'amplitude 106
2.5.3. Synthèse par modulation de fréquence 107
2.6. Synthétiser les bruits 109
2.6.1. Approximation du spectre 110
2.6.2. Transformée de Fourier inverse 110
2.6.3. Technique *overlap-add* 111
2.7. Bibliographie 111

Chapitre 3. Techniques de spatialisation des sons 119
Véronique LARCHER, Arnaud LABORIE, Rémy BRUNO, Sébastien MONTOYA

3.1. Introduction. 119
3.2. Rôle de la spatialisation dans la composition musicale 120
3.2.1. Mise en espace « acoustique » 120

3.2.2. Apport de l'électro-acoustique et de l'informatique sur la spatialisation 121
3.3. Perception de l'incidence de sources sonores dans l'espace 123
3.3.1. Les indices acoustiques de localisation 123
3.3.2. Indices de localisation et techniques de spatialisation 127
3.4. Les techniques binaurales et transaurales 127
3.4.1. Encodage des indices de localisation 127
3.4.2. Décodage pour une écoute au casque ou sur haut-parleurs. 129
3.4.3. Principales applications musicales 130
3.4.4. Performances et limites 131
3.5. Les techniques stéréophoniques et les potentiomètres panoramiques . . 133
3.5.1. Principe d'encodage et de restitution 133
3.5.2. Système d'enregistrement 134
3.6. L'holophonie ou *Wave Field Synthesis* 135
3.6.1. Modèle de représentation 136
3.6.2. Principe d'enregistrement et de restitution 137
3.6.3. Echantillonnage, fréquence limite et aliasing spatial 138
3.6.4. Réduction du nombre de sources secondaires 139
3.7. Les systèmes Ambisonic 141
3.7.1. Modèle de représentation initial : les harmoniques sphériques . . 141
3.7.2. Généralisation du modèle de représentation : les fonctions de Fourier-Bessel 144
3.7.3. Principe d'enregistrement 146
3.7.4. Synthèse 147
3.7.5. Traitement 148
3.7.6. Principe de restitution 149
3.7.7. Ambisonic et holophonie 151
3.8. Conclusion 151
3.9. Bibliographie 152

Chapitre 4. Ordonnancement pour les systèmes musicaux temps réel . . . 157
Yann ORLAREY

4.1. Introduction. 157
4.2. Spécification du problème 158
4.2.1. Définitions 159
4.2.2. Opérations sur l'ordonnanceur 159
4.2.3. Coût d'ordonnancement 160
4.2.4. Utilisation de l'ordonnanceur 160
4.3. L'algorithme d'ordonnancement 161
4.3.1. L'histoire de Monsieur Nay et de la société DO-IT 161
4.3.2. Implémentation de l'algorithme de Nay. 163
4.4. Performances 170

4.4.1. Contexte des mesures 170
4.4.2. Performances pour des séjours courts 171
4.4.3. Performances pour des séjours moyens 172
4.4.4. Performances pour des séjours longs 173
4.5. Conclusion 173
4.6. Bibliographie 174

Chapitre 5. MidiShare : une architecture logicielle pour la musique 175
Yann ORLAREY, Dominique FOBER et Stéphane LETZ

5.1. Introduction 175
5.2. MidiShare : une architecture logicielle pour la musique 176
5.2.1. Le modèle conceptuel 177
5.2.2. Des mécanismes sophistiqués pour la gestion du temps 178
5.2.3. Une boite à outils canonique 179
5.3. Exemples 183
5.3.1. Un squelette typique 183
5.3.2. Un écho 185
5.3.3. Un séquenceur 188
5.4. Quelques difficultés de la programmation temps réel 190
5.4.1. Composer avec la latence 190
5.4.2. Dérive temporelle 192
5.5. Conclusion 193
5.6. Bibliographie 194

Chapitre 6. Grammaires, automates et musique 195
Marc CHEMILLIER

6.1. Introduction 195
6.2. La hiérarchie de Chomsky 195
6.2.2. Equivalence entre grammaires et automates 197
6.2.3. Tables de transition et chaînes de Markov 198
6.2.4. Transductions rationnelles 200
6.2.5. Notion de *parsing* 201
6.3. Grammaires formelles et musique 202
6.3.1. L'article de Curtis Roads 202
6.3.2. Deux automates musicaux 204
6.4. Grammaire de substitutions de jazz 208
6.4.1. Règles de Steedman 208
6.4.2. Moteur d'application des règles 211
6.4.3. Arrangement des grilles pour piano 214
6.4.4. Analyse par automate fini avec Lex 216
6.5. Automate sériel 221
6.5.1. Régularité du langage sériel 221

6.5.2. Programme d'analyse par automate fini 223
6.6. Bibliographie 227

Chapitre 7. Programmation visuelle pour la composition 231
Gérard ASSAYAG et Carlos AGON

7.1. Introduction 231
7.1.1. Modèles de langages à objet 233
7.1.2. Modèles de langages visuels 234
7.1.3. Le langage visuel Max 234
7.2. Le langage visuel d'OpenMusic 235
7.2.1. Spécification lexicale 235
7.2.2. Spécification syntaxique 237
7.2.3. Sémantique opérationnelle 238
7.3. Implémentation 239
7.3.1. Partie statique 240
7.3.2. Protocole dynamique 241
7.4. L'environnement OM 243
7.4.1. *Workspaces* 243
7.4.2. Packages 244
7.4.3. Classes 244
7.4.4. Sous-classage 245
7.4.5. Patches, factories, instances, éditeurs 246
7.4.6. Fonctions génériques, méthodes 247
7.4.7. Editeurs musicaux 248
7.5. Maquettes 249
7.6. Quelques exemples musicaux 257
7.6.1. Approche fonctionnelle 257
7.6.2. Approche par objets 259
7.7. Conclusion 263
7.8. Bibliographie 265

Chapitre 8. Modèles temporels pour la synthèse musicale 269
Myriam DESAINTE-CATHERINE

8.1. Introduction 269
8.2. Les systèmes à base de pistes 271
8.2.1. Un extrait d'une partie instrumentale 271
8.2.2. Plusieurs parties instrumentales 273
8.2.3. Utilité pratique 274
8.3. Les systèmes hiérarchiques 275
8.3.1. Combinaison temporelle 275
8.3.2. Exemples 276
8.3.3. Edition hiérarchique 276

8.4. Les systèmes fonctionnels hiérarchiques 277
8.4.1. Représentation des structures musicales 277
8.4.2. Opérateurs structuraux 279
8.4.3. Représentations graphiques des structures musicales 280
8.4.4. Extraction d'informations à partir des termes fonctionnels 281
8.4.5. Structures musicales et formes musicales 284
8.4.6. Discussion 285
8.4.7. Les opérateurs rythmiques d'OpenMusic 292
8.5. L'approche logique 294
8.5.1. Le paradigme logique 295
8.5.2. La maintenance de relations temporelles 296
8.5.3. Limitations dans le cadre musical 297
8.6. Les systèmes hiérarchiques basés sur les grammaires 299
8.6.1. Sémantique temporelle de hiérarchies musicales 299
8.6.2. Evaluation 302
8.6.3. Grammaires attribuées relationnelles 305
8.6.4. Limitations du formalisme 306
8.7. Conclusion 307
8.8. Bibliographie 308

Chapitre 9. Algorithmes et techniques pour l'analyse musicale 311
Emilios CAMBOUROPOULOS et Pierre-Yves ROLLAND

9.1. Introduction 311
9.2. Frontières locales 313
9.3. Accents locaux et structure métrique 315
9.4. Patterns musicaux 318
9.4.1. Un algorithme d'induction de patterns basée sur des répétitions exactes 321
9.4.2. Un algorithme d'induction de patterns basée sur des répétitions approximatives 322
9.5. Segmentation 328
9.6. Similitude musicale et catégorisation 331
9.7. Conclusion 334
9.8. Annexes 334
9.8.1. L'algorithme MDFL (version améliorée) 334
9.8.2. L'algorithme Unscramble 335
9.8.3. L'algorithme FlExPat 337
9.9. Bibliographie 338

Chapitre 10. Harmonisation musicale automatique et programmation par contraintes 291
François PACHET et Pierre ROY

10.1. De l'harmonisation à la résolution de problèmes 343
10.1.1. La musique tonale et les traités d'harmonie 344
10.1.2. Les règles d'harmonie 345
10.1.3. La composition automatique 348
10.1.4. Le problème de la combinatoire 348
10.2. Harmonisation automatique sous contraintes 350
10.2.1. Les premières tentatives de modélisation du problème par les contraintes 350
10.2.2. Utilisation de la satisfaction de contraintes 351
10.2.4. Eléments algorithmiques 354
10.2.5. Les techniques de recherche locale 356
10.2.6. Autres approches 357
10.2.7. Les systèmes existants 359
10.3. Conclusion : le problème est-il résolu ? 359
10.4. Bibliographie 361

Chapitre 11. L'analyse de Fourier 365
Sylvain MARCHAND

11.1. La transformée de Fourier 367
11.1.1. Notations 367
11.1.2. Définitions 367
11.1.3. Spectres 369
11.1.4. Modèles sinusoïdaux 371
11.1.5. Transformée de Fourier à court terme 372
11.2. Imprécisions de la transformée de Fourier classique 374
11.2.1. Imprécision en fréquence 377
11.2.2. Imprécision en amplitude 378
11.2.3. Imprécision en phase 379
11.2.4. Choix de la fenêtre d'analyse 380
11.3. Méthodes pour l'amélioration de la précision 386
11.3.1. Technique de l'interpolation parabolique 386
11.3.2. Utilisation des dérivées du signal 389
11.4. Bibliographie 399

Chapitre 12. MPEG, une norme pour la compression, la structuration et la description du son 403
Rémi RONFARD

12.1. Introduction 403

12.2. Bases psycho-acoustiques et algorithmiques 404
12.2.1. Quantification et compression 405
12.2.2. Psycho-acoustique 405
12.2.3. Décomposition en sous-bandes 407
12.3. Les formats MPEG-1 et MPEG-2 409
12.3.1. MPEG-1 Layer 1 410
12.3.2. MPEG-1 Layer 2 411
12.3.3. MPEG-1 Layer 3 411
12.3.4. MPEG-2 *Advanced Audio Coding* 412
12.4. Traitement et analyse dans le domaine de compression 413
12.4.1. Traitements et effets 413
12.4.2. Analyse et indexation 413
12.5. MPEG-4 et les objets sonores 414
12.5.1. Description de scènes audio (MPEG-4 Audio BIFS) 415
12.5.2. MPEG-4 *General Audio* 415
12.5.3. MPEG-4 *Structured Audio* 415
12.5.4. MPEG-4 *Text to speech* 417
12.6. MPEG-7 et la description des sons et de la musique 417
12.6.1. Métalangage de définition des descripteurs MPEG-7 418
12.6.2. Descripteurs MPEG-7 419
12.6.3. Schémas de description MPEG-7 420
12.7. Conclusion 421
12.8. Bibliographie 421

Chapitre 13. Les normes MIDI et MIDIFiles 291
Stéphane LETZ

13.1. Les normes MIDI et MIDIFiles 423
13.1.1. Historique 423
13.1.2. Présentation de la norme MIDI 424
13.1.3. Extensions et sous-normes 426
13.1.4. La norme MIDIFile 427
13.1.5. *Parsing* d'un flot MIDI 428
13.2. Représentation du temps et synchronisation 433
13.2.1. Représentation du temps 433
13.2.2. Synchronisation 435
13.3. Limitations et évolution 436
13.4. Bibliographie 437

Index 439

Informatique Musicale
Du signal au signe musical

Introduction

François Pachet & Jean-Pierre Briot

La musique a toujours été un terrain d'expérimentation privilégié de l'informatique. Les raisons en sont multiples, mais la principale est peut être que la musique entretient des rapports serrés avec les mathématiques, à la fois du discret et du continu. En effet, la discrétisation de la musique tonale (l'échelle dite tempérée aujourd'hui) depuis le XVIIe siècle permet d'employer de nombreux modèles mathématiques qui se prêtent aisément à l'informatisation. De même, la compréhension physique des structures vibrantes est à la base de la théorie spectrale de Fourier qui est au centre de la physique des signaux musicaux. Ce rapport dialectique entre discret et continu se transpose directement en musique avec l'opposition signe (la note, la partition) et signal (physique), d'où le sous-titre de ce livre.

Ainsi des applications musicales de l'informatique ont vu le jour avec l'apparition des premiers ordinateurs. On cite souvent les fameuses compositions de Hiller et Isaacson, réalisées entièrement par ordinateur en 1957 (Hiller et Isaacson, 1959), à l'aide d'un programme comprenant quelques règles simples d'harmonie. Plus tard et en France, Pierre Barbaud reprends ces techniques et développe des automates musicaux avec lesquels il compose des jingles et musiques de films (Barbaud, 1966).
Mais l'application de l'informatique à la musique ne se limite pas à la réalisation d'automates. Aujourd'hui, toute la gamme des usages de la musique fait appel, voir relève directement, de l'informatique : de la composition à la synthèse sonore, de la spatialisation aux systèmes musicaux interactifs, de la compression à la distribution de musique par réseaux.

Ce livre n'a pas pour but de proposer un panorama exhaustif de la discipline. Celle-ci est déjà trop complexe pour se prêter à une telle entreprise. On peut citer néanmoins le *Computer Music Tutorial* de Curtis Roads (1996), qui présente un panorama relativement complet, mais qui malgré sa taille, ne peux traiter des divers sujets véritablement en profondeur.

L'objet de ce livre est plutôt de montrer que l'informatique musicale n'est pas – n'est plus – une simple collection de travaux plus ou moins isolés, mais une discipline à part entière. Même si l'informatique musicale est déjà fortement structurée en sous-domaines, nous avons choisi de la présenter de manière continue, par un parcours sillonnant le vaste chemin qui mène du signal physique au signe musical. Nous avons ainsi identifié un nombre restreint de thèmes (9) qui se prêtent particulièrement bien à une description pédagogique, de niveau ingénieur, et qui peuvent être vus comme des points d'entrée dans ce vaste paysage. Chacun de ces thèmes est suffisamment stabilisé pour faire l'objet de cours (DESS, DEA ou équivalent).

Ces thèmes sont classés en fonction de leur position dans le parcours « signal » « signe » qui n'est bien sûr qu'un parcours possible dans la discipline.

Ainsi, le chapitre 1 (analyse) traite de l'analyse des signaux audio par des techniques spectrales (Fourier). Ces analyses sont souvent la base de toutes les applications ayant à manipuler des signaux musicaux, que ce soit pour l'analyse de contenus (indexation) ou pour les applications temps réel.
Le chapitre 2 traite de la synthèse, domaine particulièrement spectaculaire, et qui monte en puissance, avec la disparition progressive des synthétiseurs « hardware » analogiques et l'importance nouvelle des cartes sons et autres périphériques audio pour les jeux vidéo.
Le chapitre 3 traite du problème de la spatialisation de sources sonores, qui connaît aujourd'hui de nombreuses applications, en particulier avec l'apparition de la norme 5.1 utilisée entre autres pour le *home cinéma*.
Le chapitre 4 traite des problèmes particuliers posés par le temps réel en informatique musicale. Dans ce domaine lui-même très riche, nous nous sommes concentrés sur les problèmes d'ordonnancement (*scheduling*), et du rapport entre langages haut-niveau et ordonnancement bas niveau, prenant l'exemple de MidiShare comme architecture canonique pour traiter ce problème.
Le chapitre 5 traite des problèmes algébriques liés à la modélisation de langages musicaux par grammaire et automates. Ce domaine, qui existe depuis les toutes premières applications de l'informatique (les automates de Hiller et Isaacson ou de Barbaud) reste encore aujourd'hui d'actualité avec le besoin pressant d'outils de génération musicaux de plus en plus évolués.
Le chapitre 6 traite de la notion de « problème musical » (et sa résolution), particulièrement important en composition assistée par ordinateur.
Le chapitre 7 traite des représentations symboliques du temps en musique, cruciales pour toutes les applications à la fois analytiques et génératives.
Le chapitre 8 présente les techniques principales pour la détection de patterns dans des représentations symboliques de la musique (partitions, Midi).
Enfin le chapitre 9 traite des techniques de programmation par contraintes qui sont de plus en plus utilisées pour la génération de musique de différents styles.

En outre, 3 chapitres plus techniques traitent de problèmes techniques généraux à l'informatique musicale (et à d'autres domaines).

Chapitre 10 : La transformée de Fourrier.
Chapitre 11 : Les formats de représentations de la famille Mpeg.
Chapitre 12 : Le format Midi et les MidiFiles.

Le choix de ces thèmes est justifié en particulier par les demandes fréquentes d'informations sur ces sujets de la part des étudiants, et par leur utilité centrale dans les applications récentes de l'informatique musicale.

Références générales sur l'informatique musicale
Une bibliographie complète des livres traitant de l'informatique musicale prendrait probablement un livre entier. Un tel effort a d'ailleurs été entrepris mais non terminé par Davis dans (Davis, 1990). Nous citons ici quelques ouvrage, la plupart en anglais, qui peuvent donner des informations techniques sur l'informatique musicale en général. Les chapitres de ce livre renvoient vers des publications plus spécialisées.
Les éditions *MIT Press* et *Swets & Zeitlinger* entre autres proposent régulièrement de nouveaux titres de qualité dans différents domaines de l'informatique musicale.

Balaban, Mira, Ebcioglu, Kemal and Otto Laske. 1992. *Understanding Music with Artificial Intelligence*. MIT Press.
Barbaud, Pierre. 1966. *Initiation à la Composition Musicale Automatique*. Dunod, Paris.
Blauert, Jens. 1983. *Spatial Hearing*. MIT Press.
Chemillier, Marc & Pachet, François, 1998. *Recherches et Applications en Informatique Musicale*, Hermes, Paris.
Cope, David. 1991. *Computers and musical style*. A-R Editions.
Cope, David. 2000. *The Algorithmic Composer*. A-R Editions.
Davis, Deta S. 1990. *A Computer Music Bibliography*. A-R Editions (2 volumes).
De Poli, Giovanni, Piccialli, Aldo and Curtis Roads. 1991. *Representations of Musical Signals*. MIT Press.
Hiller, Lejaren, & Leonard Isaacson. 1959. *Experimental Music: Composition with an Electronic Computer*. McGraw-Hill, New York.
Howell, Peter, West, Robert and Ian Cross. 1991. *Representing Musical Structure*. Academic Press.
Roads, Curtis. 1996. *Computer Music Tutorial*, MIT Press.
Roads, Curtis. 1989. *The music machine: selected readings from "Computer Music Journal"*, MIT Press.
Roads, Curtis and John Strawn. 1987. *Foundations of computer music*. MIT Press.
Roads, Curtis, Pope, Stephen Travis, Piccialli, Aldo and Giovanni De Poli. 1997. *Musical Signal Processing - Studies on New Music Research 2*, Swets & Zeitlinger.
Rowe, Robert. 1993. *Interactive Music Systems, Machine Listening and Composing*, MIT Press.
Rowe, Robert. 2001. *Machine Musicianship*, MIT Press.
Schwanauer, Stephan & David Levitt. 1993. *Machine Models of Music*.
Todd, Peter and David Gareth Loy. 1992. *Music and Connectionism*. MIT Press.

Chapitre 1

Méthodes d'analyse du signal musical

1.1. Introduction

L'un des objectifs de l'analyse du signal musical est d'en fournir une description aussi complète que possible. Les méthodes d'analyse sont très nombreuses et très diverses et permettent d'accéder, de façon plus ou moins robuste, aux divers paramètres de cette description.

Dans quel but effectuer une analyse du signal musical ? Nous en distinguerons trois. Tout d'abord, reproduire le signal musical par une machine après d'éventuelles transformations. Cela concerne l'analyse-synthèse ou le codage, l'objectif de l'analyse étant d'obtenir une représentation du signal lui-même qui autorise de telles transformations. Cela concerne aussi la synthèse et son contrôle, l'objectif de l'analyse étant d'estimer des paramètres utiles à la synthèse ou rendant son contrôle automatique.

Ensuite, permettre la réinterprétation du signal musical par un humain (ou par une machine MIDI). Il s'agit de transcrire le signal sous forme de partition (ou en signal MIDI), et donc d'obtenir une représentation de la notation musicale.

Finalement, accroître nos connaissances sur la production, la perception ou la modélisation des signaux musicaux. Il s'agit par exemple de concevoir et de valider des modèles de signaux ou encore d'établir des règles d'interprétation.

Chapitre rédigé par Boris DOVAL.

Examinons la notation musicale. C'est une forme de description de la musique, mais les musiciens savent qu'elle n'est que schématique et ne permet pas, seule, d'assurer une production de qualité. Le musicien est appelé à « interpréter » cette notation. Fournir une description du signal musical nécessite donc d'associer à la notation musicale des informations sur la façon dont le son est produit.

Explicitons donc en quoi la notation musicale décrit le signal. Elle est constituée principalement de notes définissant la fréquence et la durée relative du son à produire, d'indications d'articulation (liaisons, piqués, etc.) associées à la forme de l'enveloppe temporelle de chaque note, de nuances définissant, sur une échelle sommaire, la force du son, et du tempo permettant de relier les durées relatives et les durées absolues.

Parallèlement, la seule indication sur la production du son est le type d'instrument qui détermine un grand nombre de paramètres souvent rassemblés sous la notion de « timbre » : son percussif ou entretenu, enveloppe spectrale des partiels, évolution de cette enveloppe, présence d'inharmonicités, de bruits transitoires ou entretenus, de non-linéarités, etc.

Enfin, le jeu de l'instrumentiste n'est en général pas précisé non plus : par exemple le vibrato, les attaques, la tenue des sons (sons filés, soufflets), bref, tout ce qui fait la « vie » d'une note.

De cet examen rapide de la notation musicale se dégagent les analyses qui permettent de décrire le signal musical. Pour représenter le signal sonore, il faut savoir estimer l'enveloppe temporelle, les partiels, la fréquence fondamentale, l'enveloppe spectrale des partiels, la partie non périodique, et pouvoir suivre leurs évolutions dans le temps. Pour analyser les paramètres de jeu du musicien, il faut savoir en déduire les caractéristiques de l'enveloppe temporelle de chaque note, du vibrato (fréquence et amplitude) et d'autres variations du fondamental. Enfin, la notation musicale nécessite de savoir segmenter la phrase musicale en notes, estimer la durée, la fréquence et la force de chaque note, extraire le tempo (battue/rythme).

Ce chapitre ne prétend pas fournir des réponses à tous les problèmes évoqués ci-dessus, mais présente des méthodes d'analyse qui s'articulent autour de systèmes d'analyse-synthèse ou de « MIDIfication » du signal acoustique. Il est conçu pour que le lecteur puisse programmer les techniques élémentaires d'analyse permettant par exemple de « MIDIfier » un signal musical monophonique, tout en donnant des références de méthodes plus performantes. Les prérequis nécessaires sont les notions élémentaires de traitement du signal et d'algorithmique de niveau deuxième cycle universitaire. Les ouvrages suivants peuvent être consultés en complément : [BLA 98, PIC 98, RAB 75].

1.2. Outils d'analyse du signal sonore

1.2.1. *Le signal musical et sa numérisation*

Pour analyser le signal musical, il est nécessaire de connaître les ordres de grandeurs des mesures effectuées sur ce signal. En voici donc quelques grandeurs caractéristiques.

L'intensité I exprimée en dB SPL (*sound pressure level*) correspond à la mesure en décibels du carré de la variation de pression acoustique p_a prise en un point rapportée à la pression de référence $P_{\text{ref}} = 2 \times 10^{-5}$ Pa, c'est-à-dire $I = 10 \log_{10}((p_a/P_{\text{ref}})^2) = 20 \log_{10}(p_a/P_{\text{ref}})$. Les sons produits par les instruments ou par la voix ont une intensité acoustique pouvant varier sur une étendue d'environ 120 dB SPL, bien que cette étendue soit rarement utilisée ou même atteinte par un seul instrument.

La hauteur des notes est mesurée par leur fréquence exprimée en hertz. L'intervalle musical IM séparant deux notes de fréquences f_1 et f_2 est exprimé en demi-tons tempérés par $IM = 12 \log_2(f_1/f_2)$ où $\log_2(x) = \log(x)/\log(2)$. L'intervalle d'octave correspond à un rapport 2 entre les fréquences. Les sons produits par des instruments à hauteur de note définie couvrent une étendue de fréquence pouvant être très grande : de 27,5 Hz (période = 36 ms) à 4 186 Hz (0,239 ms) pour le piano, de 65,4 Hz (15 ms) à 349 Hz (2,86 ms) pour une tessiture de basse Do1-Fa3, de 261,6 Hz (3,82 ms) à 1 046 Hz (0,95 ms) pour une tessiture de soprano Do3-Do5.

Cependant, lors de la production d'une note à une fréquence F_0, sont aussi produites des fréquences plus élevées. Lorsque le son est périodique, ces fréquences sont multiples de F_0 et sont appelées harmoniques. Le nombre d'harmoniques d'amplitude significative varie beaucoup selon l'instrument et la hauteur de la note : une ou deux pour un sifflement, moins d'une dizaine pour la flûte, plusieurs dizaines pour le piano dans le grave.

Les analyses présentées ci-après font toutes l'hypothèse que le signal analysé est la variation de tension issue d'un capteur de type microphone mesurant les variations de pression acoustique en un point. De plus, le signal étant traité par algorithme, il est converti en un signal numérique noté $s(k)$, $k = 0, \ldots, K-1$ [1], à la fréquence d'échantillonnage $F_e = 1/T_e$. Ce signal est donc à bande limitée $[-F_e/2, F_e/2]$. Comme il est supposé réel, la bande de fréquence « utile » à analyser se restreint à $[0, F_e/2]$. De plus, chaque échantillon est codé sur un nombre limité de bits, ce qui limite la dynamique et introduit un bruit de quantification. La dynamique augmentant

1. Dans toute la suite, nous utiliserons la convention suivante (utilisée notamment en langage C) pour les indices : les indices d'un tableau de taille N vont de 0 à $N-1$.

de 6 dB par bit, les meilleurs systèmes d'enregistrement utilisent 20 bits pour couvrir la dynamique de 120 dB, mais les plus courants 16 pour 96 dB de dynamique (CD audio, DAT).

Les fréquences audibles couvrent la bande de 20 Hz à 20 000 Hz. La fréquence d'échantillonnage du CD audio, $F_e = 44\,100$ Hz, permet donc de représenter l'ensemble des fréquences audibles. Cependant, dans de nombreuses analyses, la bande des fréquences utiles est nettement inférieure et des fréquences d'échantillonnage de 16 000 Hz, 8 000 Hz ou même inférieures seront utilisées pour réduire le nombre d'échantillons à traiter.

1.2.2. *Analyse à court terme dans le domaine temporel*

La musique s'inscrit dans le temps. Elle est une succession d'événements généralement distincts à des échelles de temps très diverses, allant d'une milliseconde à une vingtaine de millisecondes pour les périodes fondamentales, d'un dixième de seconde à plusieurs secondes pour la durée des notes. La plupart des analyses étudient donc le signal localement avec une échelle temporelle adaptée de façon à caractériser ces événements séparément.

L'analyse à court terme permet de se focaliser sur une partie du signal concentrée dans le temps. Pratiquement, il suffit de multiplier le signal par une fonction nulle en dehors d'un support temporel borné. Une telle fonction, notée $w(k)$, $k = 0, \ldots, M-1$, est appelée « fenêtre de pondération ». Les paramètres importants sont la taille (M échantillons de signal correspondant à une durée de MT_e secondes) et le type de la fenêtre (rectangulaire, Hanning, Hamming, Blackmann, Blackmann-Harris, Kaiser, etc.). La fenêtre de Hanning est définie par $w(k) = 0{,}5 - 0{,}5\cos(2\pi k/M)$, celle de Hamming par $w(k) = 0{,}54 - 0{,}46\cos(2\pi k/M)$, celle de Blackmann par $w(k) = 0{,}42 - 0{,}50\cos(2\pi k/M) + 0{,}08\cos(4\pi k/M)$. Plus la taille de la fenêtre sera grande, plus l'analyse sera « délocalisée ». Le type de fenêtre a une influence sur l'étalement fréquenciel provoqué par le fenêtrage (voir paragraphe 1.2.3).

Si, en théorie, l'analyse peut être effectuée à chaque instant, en pratique elle est effectuée à des instants particuliers appelés « instants d'analyse » et notés t_l, $l = 0, \ldots, L-1$. La durée entre deux instants d'analyses successifs est donc un paramètre supplémentaire qui règle la précision temporelle de l'analyse. Dans le cas où les instants d'analyses sont régulièrement répartis, cette durée est constante et s'appelle période d'analyse T_a. Son inverse est la fréquence d'analyse F_a. Le nombre de points entre deux fenêtres successives est $D = T_a/T_e$. Dans le cas contraire, les instants d'analyse sont en général déterminés par les résultats d'une autre analyse. C'est le cas de l'analyse synchrone à la période fondamentale, utilisée notamment pour certaines transformations de durée et de hauteur de la voix, où les instants d'analyse correspondent à un point particulier d'une période fondamentale du signal.

Il est souvent nécessaire d'attribuer un instant précis à l'analyse effectuée sur une fenêtre. Dans la mesure où les fenêtres sont presque toujours symétriques, et pour respecter la régularité des instants d'analyse, l'instant précis d'analyse est très souvent choisi au centre de la fenêtre : $t_l = (lD + (M-1)/2)T_e$. Notons que dans certaines applications, le premier instant doit être à zéro, auquel cas le signal est complété par des zéros pour les temps négatifs.

Finalement, l'analyse à court terme consiste à construire la suite des signaux s_l à support temporel borné, appelés « trames » et définis par :

$$s_l(k) = w(k)s(lD + k) \qquad \text{pour } k = 0, \ldots, M-1$$

où $l = 0, \ldots, L-1$ désigne le numéro de la trame. Remarquons que la durée entre deux instants d'analyses successifs doit être inférieure à la taille de la trame pour éviter d'« oublier » certains échantillons de signal, ce qui est assuré dans le cas d'une analyse régulière par $D \leqslant M$.

1.2.3. *Analyse à court terme dans le domaine fréquenciel*

La structure de la musique se fonde aussi et surtout sur la notion essentielle de fréquence. Que ce soit pour déterminer la hauteur des notes, leur contenu harmonique ou la répartition de l'énergie par bande de fréquence, l'outil d'analyse fréquencielle est très utile.

Effectuer une analyse à court terme dans le domaine fréquenciel consiste simplement à appliquer une transformée de Fourier (TF) sur chaque trame issue de l'analyse précédente. En fait, pour qu'elle soit calculable, la transformée utilisée est la transformée de Fourier discrète (TFD), calculée par un algorithme de transformée de Fourier rapide (TFR, en anglais FFT). Le résultat est une représentation temps-fréquence qui est appelée transformée de Fourier à court terme (TFCT) [ALL 77, POR 80] et qui décrit le contenu fréquenciel du signal à chaque instant d'analyse :

$$\widetilde{s_l}(n) = \mathrm{TFD}(s_l)(n) = \sum_{k=0}^{N-1} e^{-j2\pi nk/N} s_l(k)$$

pour $n = 0, \ldots, N-1$.

Aux N points de signal, la TFD associe N points de spectre complexe que l'on représente habituellement sous forme polaire (module et phase). Ces N points, numérotés de 0 à $N-1$, correspondent à la bande de fréquence $[0, F_e[$. Le signal étant réel, le spectre est à symétrie hermitienne et seuls les points $[0, N/2]$ sont utiles. Le n-ième point de TFD correspond donc à la fréquence $f = nF_e/N$.

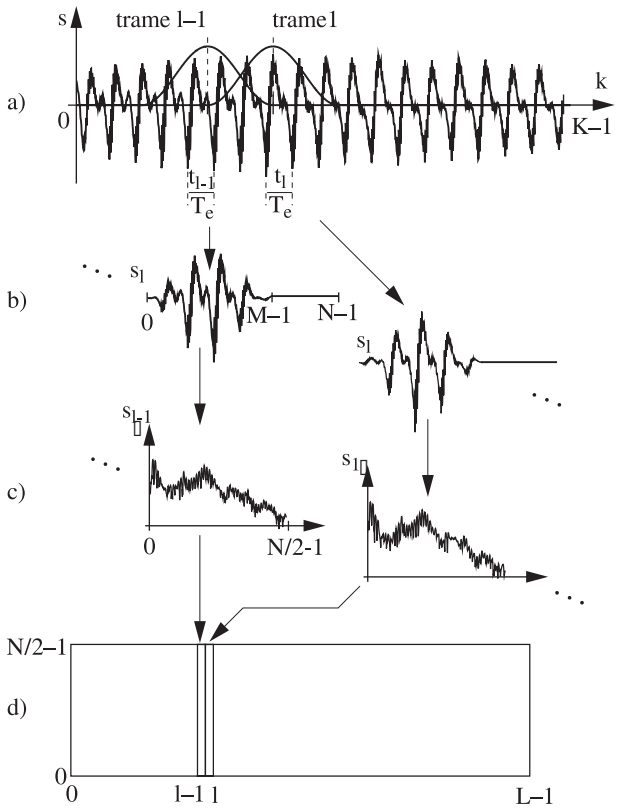


Figure 1.1. *Représentations du signal : a) représentation temporelle, b) représentation temporelle à court terme, c) représentation fréquencielle à court terme, d) spectrogramme*

Quel que soit N, les points de TFD représentent toujours la même bande de fréquences. Le rapport N/M agit donc comme un facteur de « suréchantillonnage » de la TF. Remarquons que dans l'équation précédente, la TF est prise sur N points de signal, bien que la fenêtre soit de taille M. En pratique, cela revient à compléter le signal fenêtré par des zéros (*zero padding* en anglais). Finalement, pour le calcul de la TFCT, il faut effectuer les choix de la taille et du type de fenêtre, de la fréquence d'analyse et de la précision spectrale d'affichage, ce qui revient à choisir M, D et N.

Le choix de M est un compromis : suffisamment grand pour englober le phénomène étudié et suffisamment petit pour ne pas intégrer ses variations dans une même trame. La quantité $1/MT_e$ est la résolution intrinsèque de l'analyse spectrale.

Le choix de N est moins critique : il doit être supérieur ou égal à M, suffisamment grand pour obtenir une bonne précision spectrale, mais limité par le temps de calcul (en $N \log(N)$) et par les algorithmes de TFR les plus courants qui imposent que N soit une puissance de 2. La quantité $1/NT_e$ est la précision d'affichage de l'analyse spectrale.

De son côté, le choix de $T_a = DT_e$ doit permettre d'« échantillonner » le phénomène étudié suffisamment finement pour suivre ses variations. Ainsi, d'après le théorème de Shannon, au-dessus d'une certaine valeur de T_a, l'analyse ne traduit pas les variations rapides du phénomène, par contre en dessous de cette valeur, diminuer T_a fournit plus de points d'analyse mais pas plus de précision dans les variations. Il y a donc une valeur optimale qui dépend de la taille et du type de fenêtre. Une heuristique [ALL 77] consiste à faire en sorte que les points soient pondérés également par la superposition, d'où le choix de $D = M/P$, où P est la largeur du lobe principal de la fenêtre de M points, exprimée en points de FFT (voir tableau 1.1).

Finalement, le choix de la fenêtre de pondération est dicté par des considérations spectrales. Les fenêtres sont utilisées pour réduire l'étalement spectral et les rebonds dus à la troncature temporelle. La TF de la fenêtre de pondération étant constituée d'un lobe principal et de lobes secondaires, il faudra faire en sorte que la largeur du lobe principal soit la plus faible possible, que l'amplitude des plus hauts lobes secondaires soit la plus faible possible, et enfin, selon les cas, que la pente des lobes secondaires soit la plus raide possible. La largeur du lobe principal exprimée en hertz est inversement proportionnelle à MT_e, le coefficient de proportionnalité (en points, voir tableau 1.1) dépendant du type de fenêtre. Par contre, elle est indépendante de N, ce qui montre bien qu'augmenter N améliore la précision de l'affichage (il y a plus de points) mais ne change en rien la résolution spectrale. L'amplitude et la pente des lobes secondaires, elles, ne dépendent que du type de fenêtre. Ceci implique notamment qu'agrandir la fenêtre ne réduit pas l'influence des lobes secondaires. Le choix de la fenêtre est alors un compromis entre ces critères.

Le tableau 1.1 résume les propriétés de quelques fenêtres parmi les plus utilisées. Une étude très détaillée peut être trouvée dans un article dû à F.J. Harris [HAR 78].

Il faut remarquer que la représentation graphique du module en dB de la TFCT, $20 \log_{10} |\widetilde{s_l}(n)|$ sous la forme d'une image de niveaux de gris, appelée spectrogramme, est très utilisée pour l'analyse visuelle des sons. Les temps discrets, $l = 0, \ldots, L-1$, sont en abscisse, les fréquences discrètes, $n = 0, \ldots, N-1$, en ordonnée et les amplitudes en dB sont représentées par les niveaux de gris de l'image. Une normalisation et un seuillage des amplitudes sont nécessaires pour une meilleure lisibilité.

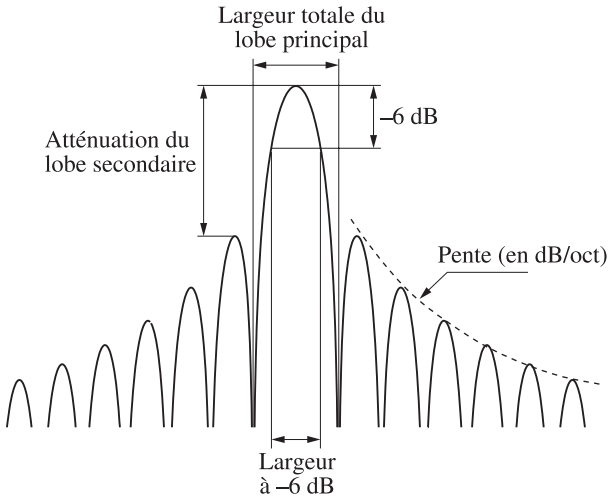


Figure 1.2. *Caractéristiques des fenêtres de pondération*

Fenêtre	**Largeur -6 dB lobe principal (pts)**	**Largeur totale lobe principal (pts)**	**Atténuation lobes second. (dB)**	**Pente (dB/Oct)**
Rectangulaire	1,21	2	-13	-6
Triangulaire	1,78	4	-26	-12
Hanning	2,00	4	-31	-18
Hamming	1,81	4	-42	-6
Blackmann	2,35	6	-58	-18
Kaiser $\alpha = 2{,}0$	1,99	4,5	-45	-6
Kaiser $\alpha = 3{,}0$	2,39	6,4	-69	-6

Tableau 1.1. *Caractéristiques de quelques fenêtres de pondération. La largeur à -6 dB (d'après [HAR 78]) et la largeur totale approximative du lobe principal (entre les deux minimums les plus proches) sont exprimées en nombre de points dans le cas où $N = M$; si $N \neq M$, les valeurs indiquées sont à multiplier par N/M.*

1.2.4. *Robustesse*

Le signal musical est extrêmement complexe et les analyses de ce signal sont souvent délicates. Il est donc important de s'assurer de la robustesse des analyses effectuées.

La notion de robustesse peut être vue comme le fait que les résultats de l'analyse seront peu sensibles à de petites variations du signal d'entrée. Pour illustrer cela, supposons que nous désirions connaître l'amplitude et la fréquence du partiel le plus fort dans le spectre du signal sonore. L'algorithme est évident : il consiste à déterminer le maximum du spectre et de prendre l'indice de ce maximum. Nous pourrons dire que l'estimation de cette amplitude est robuste, mais que celle de la fréquence correspondante ne l'est pas, car une modification infime de l'amplitude du partiel le plus fort peut modifier radicalement la fréquence du maximum, comme par exemple dans le cas de deux partiels ayant presque la même amplitude mais des fréquences très différentes.

Cependant, cette notion de robustesse est plutôt attachée à la grandeur mesurée qu'à l'analyse effectuée : ainsi, l'estimation de l'énergie est robuste, mais celle de la fréquence fondamentale ne l'est généralement pas.

1.3. Analyse de l'enveloppe temporelle

L'amplitude du signal acoustique peut être analysée à différentes échelles. La plus large qui nous concerne ici est l'échelle de la note et l'enveloppe temporelle d'amplitude traduit les variations d'intensité des différentes notes successives. Une échelle plus fine permet de mettre en évidence les variations d'intensité au cours de chaque note. Enfin, une échelle très fine révèle les variations de chaque période dans le cas d'un signal pseudo-périodique.

1.3.1. *Enveloppe temporelle*

La structure de l'enveloppe temporelle d'amplitude d'une note (on parle simplement d'enveloppe temporelle) est souvent décrite, en informatique musicale, par quatre phases successives : l'attaque, la décroissance, la tenue et le relâchement, plus connues en anglais sous les termes *attack*, *decay*, *sustain* et *release* (ADSR). Cette décomposition est notamment utilisée pour la synthèse par échantillonnage où les phases A, D et R sont reproduites telles quelles et où la phase S, la plus variable, est bouclée pour satisfaire à la durée de la note à synthétiser et comporte éventuellement des variations d'intensité. Le modèle sous-jacent est celui d'un signal périodique [2] $x_{T_0}(t)$ modulé en amplitude par cette enveloppe temporelle $A(t)$:

$$s(t) = A(t)x_{T_0}(t)$$

2. En toute généralité, il s'agit d'un signal composé d'une somme finie de sinusoïdes non nécessairement harmoniques, ce qui permet de traiter aussi les sons percussifs et/ou inharmoniques de type cloche. Il faut alors remplacer F_0 par la fréquence de la composante la plus grave (c'est-à-dire de plus petite fréquence).

Bien entendu, l'enveloppe temporelle est supposée varier lentement par rapport aux variations de x_{T_0}. L'enveloppe temporelle $A(t)$ est donc un signal à bande limitée $[-B, +B]$ telle que la largeur de bande soit inférieure à la plus petite fréquence de x_{T_0} : $2B < F_0$.

1.3.2. *Estimation de l'enveloppe temporelle*

Un des systèmes les plus simples pour estimer l'enveloppe temporelle est analogique : il s'agit du redressement simple ou double alternance suivi d'un filtrage passe-bas, réalisable par un simple pont de diode et un circuit RC. Ce principe est utilisé en radio pour effectuer une démodulation AM. Mais il est aussi utilisé en musique dans les systèmes analogiques d'analyse du signal musical.

Le seul paramètre à régler ici est la fréquence de coupure f_c du filtre passe-bas. Elle doit être choisie pour séparer au mieux le contenu fréquenciel de l'enveloppe $(A(t))$ de celui du signal périodique sous-jacent $(x_{T_0}(t))$ et devra donc appartenir à l'intervalle $[B, F_0/2]$. De plus, pour éviter des déphasages non linéaires importants, le filtre passe-bas n'aura pas une bande de transition très étroite, ce qui conduit à choisir f_c plutôt proche de B. L'ordre de grandeur de B est de 10 à 30 Hz.

Le principal défaut de cette technique est la difficulté d'estimer correctement le gain à appliquer à la sortie du filtre (supposé de gain 1 en dessous de f_c) pour que l'estimation « enveloppe » vraiment le signal. Dans le cas simple où $x_{T_0}(t)$ peut être considéré comme constitué d'une seule harmonique, il suffit de diviser le résultat par la valeur moyenne du cosinus redressé, à savoir par $1/\pi$ ou $2/\pi$ selon qu'il s'agit de simple ou de double alternance. Mais si $x_{T_0}(t)$ est riche en harmoniques, ce calcul ne convient plus et l'estimation de l'enveloppe ne sera correcte qu'à un coefficient multiplicatif près.

Une variante qui ne présente pas cet inconvénient consiste à suivre le signal dans sa phase montante et à redescendre en exponentielle décroissante de constante de temps τ [KUH 90]. La version numérique de cette technique s'écrirait alors : $\hat{A}(k) = \max(\hat{A}(k-1)e^{-T_e/\tau}, |s(k)|)$, où $s(k)$ représente l'échantillonnage de $s(t)$ à la période T_e, et $\hat{A}(k)$ l'estimation de l'enveloppe temporelle. La constante de temps τ doit être suffisamment grande pour éliminer les harmoniques de $x_{T_0}(t)$, mais suffisamment petite pour suivre la décroissance de l'enveloppe. Elle correspond, à un coefficient près, à l'inverse de la fréquence de coupure précédemment définie : $\tau = 1/(2\pi f_c)$.

L'avantage de cette technique est la fidélité de l'estimation dans les attaques de notes. Or, l'attaque de la note, qui est très importante dans la perception de l'identité des différents instruments de musique, se trouve être la partie de l'enveloppe qui évolue le plus rapidement. Cependant, le résultat présente des oscillations assez fortes qui nécessitent un filtrage passe-bas à f_c.

Bien qu'il soit tout à fait possible de réaliser ces estimations par algorithme (valeur absolue, puis filtrage passe-bas numérique), la technique numérique la plus employée consiste à estimer l'enveloppe par l'énergie à court terme du signal, qui n'est autre qu'une analyse à court terme suivie d'un calcul d'énergie sur chaque trame s_l de signal :

$$e(l) = \sum_{k=0}^{M-1} s_l^2(k) \qquad \text{pour } l = 0, \ldots, L-1$$

d'où est déduite l'estimation de l'amplitude :

$$\hat{A}_l = \sqrt{\frac{e(l)}{M}}$$

Bien entendu, la fréquence d'échantillonnage de $\hat{A}_l$ étant égale à la fréquence d'analyse F_a, il faut suréchantillonner $\hat{A}_l$ d'un facteur D pour l'aligner sur le signal.

Ici, c'est le paramètre de taille de la fenêtre qui détermine l'effet de filtrage. Plus la taille est grande, plus le signal est « moyenné » et donc filtré. En fait, ce calcul peut être interprété comme le filtrage RIF par une fenêtre rectangulaire du signal s_l^2. La largeur totale du lobe principal de la fenêtre rectangulaire étant égale à $2/(MT_e)$, la fréquence de coupure f_c correspond approximativement à $1/(MT_e)$. Il faut remarquer que ce filtrage est très médiocre en termes de performance de filtrage. Ce faisant, l'utilisation d'une fenêtre autre que la fenêtre rectangulaire doit se concevoir comme RI d'un filtre passe-bas, ce qui permet alors de réduire les oscillations résiduelles. Comme dans le cas de la première technique présentée, cette technique ne permet pas d'estimer facilement le gain à appliquer pour que l'estimation « enveloppe » vraiment le signal.

Une dernière technique consiste à estimer l'enveloppe comme le module du signal analytique correspondant à $s(t)$. En effet, le signal analytique est un signal complexe dont la partie réelle est le signal d'entrée et tel que ses parties réelle et imaginaire sont en quadrature. Ainsi, le signal analytique d'un cosinus est l'exponentielle complexe correspondante. De plus, sous l'hypothèse que $A(t) \geqslant 0$ est à bande limitée $[-B, +B]$, le signal analytique de $A(t)\cos(2\pi ft)$, où $f > B$, est le signal $A(t)e^{j2\pi ft}$ dont le module est $A(t)$. Bien entendu, dans notre hypothèse, le cosinus est remplacé par le signal périodique $x_{T_0}(t)$ qui peut s'écrire comme une somme de cosinus et le module du signal analytique est donc $A(t)$ multiplié par le module d'une somme d'exponentielle complexe, qui, cette fois-ci, n'est pas constante. Le résultat présente donc des oscillations aux fréquences du signal $x_{T_0}(t)$ qui peuvent être particulièrement importantes. Un filtrage passe-bas à une fréquence de coupure f_c (dont la valeur a été précédemment discutée) s'avère donc nécessaire. Enfin, le signal analytique se calcule soit en appliquant un filtre RIF passe-tout déphaseur pur dont on peut trouver les caractéristiques dans [RAB 74], soit en annulant les fréquences négatives de la transformée de Fourier.

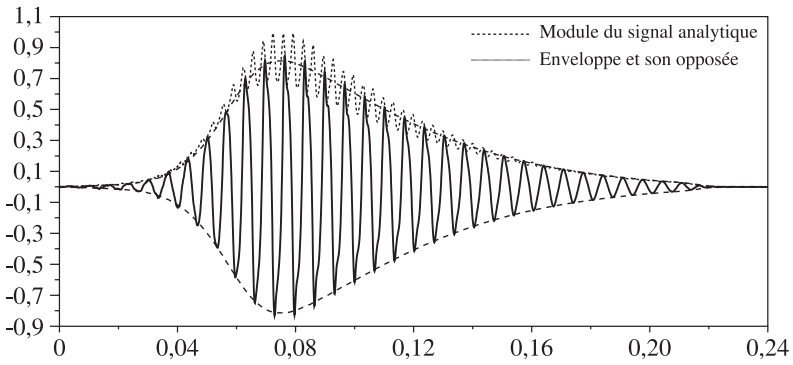


Figure 1.3. *Estimation d'enveloppe temporelle par la méthode du signal analytique. Le signal est un son court de clarinette. Le signal analytique est obtenu par la formule* $sa = \mathrm{TF}^{-1}(\widetilde{sa})$*, où* $\widetilde{sa}(n) = 2\tilde{s}(n)$ *pour* $n < M/2$*,* $\widetilde{sa}(n) = 0$ *sinon. La TF est prise sur* M *points (ici,* $M = 1\,920$*, soit 240 ms). Observez les oscillations importantes du module de* sa *(en pointillés). L'enveloppe (en tirets) a été obtenue par filtrage passe-bas (sans retard de groupe) de* sa *à la fréquence de coupure* $f_c = 15$ *Hz.*

1.3.3. *Segmentation*

Dans l'optique d'une « MIDIfication » du signal acoustique, il faut d'abord segmenter le signal, ce qui revient à détecter le début et la fin de chaque note. Cette détection peut être partiellement résolue en exploitant l'estimation de l'enveloppe temporelle.

La technique la plus directe est de seuiller l'enveloppe : lorsque l'enveloppe dépasse le seuil, cela indique un début de note ; quand elle redescend sous le seuil, cela signale une fin de note.

Malheureusement, cette technique est très sensible à la présence de bruit et est incapable de détecter le début de notes enchaînées dans les cas très courants où l'amplitude ne redescend pas sous le seuil entre deux notes successives.

En fait, l'attaque des notes est plus facilement repérable, car elle est caractérisée par une variation rapide d'amplitude. La technique la plus utilisée est donc de détecter les maximums locaux de la dérivée de l'enveloppe temporelle.

Les difficultés proviennent principalement de la présence de bruit de fond, des modulations du signal et des oscillations résiduelles de la plupart des techniques d'estimation de l'enveloppe temporelle. Ceci peut produire un grand nombre de fausses détections.

Les solutions envisagées sont d'utiliser un seuil global (en général très bas) pour éliminer le bruit et de fixer une durée minimale de la note pour éliminer les détections trop rapprochées.

1.4. Analyse des partiels et du bruit

L'oreille humaine est très sensible au contenu fréquenciel des sons. Le « timbre » des sons est souvent étudié au travers de la répartition spectrale de l'énergie et de son évolution dans le temps. Selon les catégories d'instruments de musique (à hauteur bien définie ou non, à son entretenu ou à son percussif), ce contenu fréquenciel peut être de différentes natures (partiels sinusoïdaux ou sinusoïdaux amortis, bruits de tous types).

L'analyse des partiels et du bruit repose donc sur une représentation du signal musical $s(t)$ en somme de partiels $s_i(t)$ et de bruit $b(t)$:

$$s(t) = \sum_i s_i(t) + b(t)$$

En pratique, la notion de partiel étant suffisamment large, il est tout à fait possible de représenter une partie du bruit par des partiels. Ceci laisse une certaine liberté quant à l'orientation de l'analyse : McAulay et Quatieri [MCA 86] et Fitz et Haken [FIT 96] représentent tout ce qui peut l'être dans le signal par des partiels, Serra et Smith [SER 90] sélectionnent les partiels représentatifs de la partie déterministe et modélisent le bruit séparément, Yegnanarayana *et al*. [YEG 98] ne retiennent que les partiels harmoniques et considèrent le reste comme le bruit qui contient alors toutes les apériodicités.

C'est l'application qui va orienter l'analyse : pour le codage, la technique de McAulay et Quatieri [MCA 86] convient parfaitement, mais pour l'analyse-synthèse avec modification, il est nécessaire de distinguer la partie pseudo-périodique de la partie bruit.

1.4.1. *Partiels d'un signal*

Par définition, le partiel d'un signal est une composante sinusoïdale de ce signal. Un partiel est donc caractérisé par une fréquence centrale f, une amplitude A et une phase initiale ϕ, et son expression numérique est : $s_i(k) = A\cos(2\pi fkT_e + \phi)$, $k = 0, \ldots, K-1$. Le signal complexe correspondant est $A\,e^{j(2\pi fkT_e+\phi)}$. Une définition plus générale est le partiel sinusoïdal amorti $A\,e^{-\alpha k+j(2\pi fkT_e+\phi)}$, où $\alpha > 0$ est un coefficient d'amortissement constant.

Un signal composé d'une somme de partiels a un spectre de raies. Cependant, l'observation pratique de ce signal nécessite de le tronquer, ce qui produit dans le

spectre un élargissement des raies et des oscillations. Cet effet spectral de la troncature (appelé phénomène de Gibbs) s'explique par le fait que multiplier le signal par une fenêtre de pondération (troncature temporelle) est équivalent à convoluer le spectre du signal par le spectre de la fenêtre. Ainsi, chaque raie spectrale sera remplacée, dans le spectre du signal observé, par l'image du spectre de la fenêtre à la position de la raie.

La technique d'analyse spectrale utilisée dans la suite de ce paragraphe est la TFCT. Des techniques plus précises existent mais imposent des contraintes de stationnarité qui ne sont pas respectées par le signal musical.

Les problèmes à résoudre sont alors :

1) la détection d'un partiel dans du bruit,

2) la séparation de plusieurs partiels.

Le choix des paramètres d'analyse spectrale (voir paragraphe 1.2.3 pour une discussion) doit assurer que chaque partiel présent dans le signal provoque l'apparition d'un pic sur le spectre (sinon, il sera impossible de le détecter) et que la résolution spectrale sera suffisante pour séparer deux partiels quelconques. Il faut donc que l'atténuation des lobes secondaires de la fenêtre de pondération soit supérieure au plus grand écart d'amplitude entre deux partiels et que la largeur à -6 dB du lobe principal soit inférieure au plus petit écart de fréquence entre deux partiels (voir tableau 1.1).

1.4.2. *Détection et sélection de pics*

Le principe est simple : il s'agit de repérer tous les pics du spectre d'amplitude (c'est-à-dire les maximums locaux) et de considérer qu'un pic correspond à un partiel. Bien sûr, le signal évoluant dans le temps, cette opération s'effectue à court terme, c'est-à-dire sur chaque trame de la TFCT.

L'algorithme le plus simple est donc :

Pour chaque trame $l = 0, \ldots, L-1$
  $i \leftarrow 0$
  Pour $n \leftarrow 1$ à $N-2$
    Si $|\widetilde{s_l}(n)| > |\widetilde{s_l}(n-1)|$ et $|\widetilde{s_l}(n)| > |\widetilde{s_l}(n+1)|$
      $i \leftarrow i+1$
      $h_l(i) \leftarrow n$
  $I_l \leftarrow i$

Le nombre de pics de la trame l est I_l et ces pics correspondent aux points du spectre d'indices $h_l(i)$, $i = 1, \ldots, I_l$. La fréquence en hertz du pic numéro i est $f_{l,i} = h_l(i)F_e/N$ et son amplitude complexe est $A_{l,i} = \widetilde{s_l}(h_l(i))$.

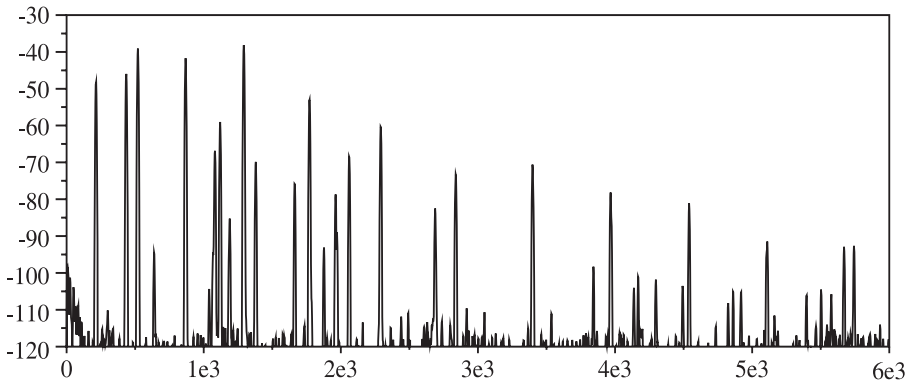


Figure 1.4. *Exemple de spectre de son de cloche. La plupart des pics représentent un partiel dont la fréquence est en abscisse et l'amplitude en ordonnée. Les partiels les plus importants ont comme fréquence 520 Hz, quelques multiples de 440 Hz et, dans les hautes fréquences, quelques multiples de 570 Hz.*

Cet algorithme extrait tous les pics du spectre. Si l'analyse prévoit une sélection, la question qui se pose est de déterminer, parmi les pics trouvés, ceux qui correspondent effectivement à un partiel et ceux qui sont des pics parasites.

Pour éliminer les pics parasites correspondant au bruit de fond, un seuil absolu (très bas ou estimé sur du « silence ») peut être utilisé.

L'atténuation des lobes secondaires de la fenêtre de pondération étant connue, tout pic dont l'amplitude relative à celle du pic le plus fort est inférieure à cette atténuation doit être rejeté. Par exemple, en utilisant la fenêtre de Hamming, tout pic dont l'amplitude est inférieure à celle du pic le plus fort moins 42 dB est à rejeter. En toute rigueur, il faudrait tenir compte de la pente de décroissance des lobes secondaires, mais la pratique est presque toujours de seuiller l'amplitude des pics relativement au pic le plus fort en choisissant ce seuil selon le type de fenêtre utilisé (voir tableau 1.1).

La sélection peut porter sur la « qualité » du pic : dans [SER 90], les pics sont sélectionnés par leur hauteur, c'est-à-dire par l'écart entre l'amplitude du pic et l'amplitude des vallées de chaque côté du pic.

Fitz et Haken interprètent leur sélection en termes de masquage psychoacoustique [FIT 96]. Pratiquement, ils partagent le spectre en bandes de fréquences logarithmiques et appliquent un seuil relatif sur chaque bande. Ceci est spécialement intéressant pour les analyses à grande F_e. Enfin, l'utilisation d'une bande de fréquences utiles est aussi une sélection.

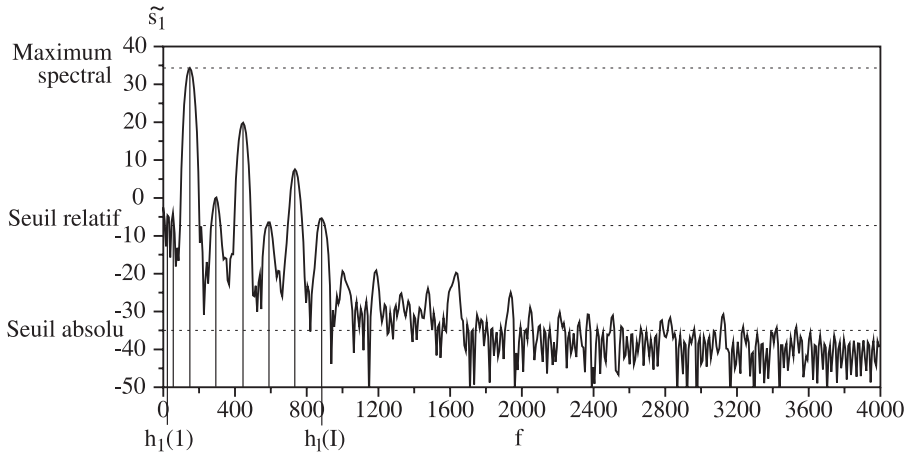


Figure 1.5. *Détection et sélection de pics spectraux. Le seuil absolu (ici -35 dB) élimine les pics correspondant au bruit de fond. Le seuil relatif au maximum spectral (ici $\max - 42$ dB) élimine les lobes secondaires (dont un exemple est visible à environ 200 Hz) et les pics trop faibles. Il reste ici $I = 8$ pics dont six sont des partiels et deux (les deux premiers) pourraient être éliminés pour leur mauvaise « qualité » (voir texte). Le spectre est calculé sur le signal de la figure 1.3.*

1.4.3. *Interpolation spectrale des pics*

L'algorithme précédent fournit des indices de la TFD comme position en fréquence des pics. Dans la plupart des cas, la précision sur les valeurs de fréquences F_e/N est très insuffisante (par exemple, $F_e = 44\,100$ Hz, $N = 1\,024$, précision $= 43$ Hz entre deux indices). La solution consistant à suréchantillonner le spectre en complétant la fenêtre de signal par des zéros lors du calcul de la TFCT (voir paragraphe 1.2.3) est inapplicable car elle augmente démesurément la complexité pour le calcul de la TFCT, mais aussi pour l'algorithme de détection des pics.

La solution qui lui est préférée consiste à interpoler le spectre uniquement au voisinage des pics. Le point de vue le plus direct effectue une interpolation polynômiale de degré 2 ou 3, utilisant donc trois ou quatre points répartis autour du pic à estimer, et renvoie la valeur de fréquence et d'amplitude du maximum du polynôme. Les formules pour l'interpolation parabolique sont les suivantes : soit le i-ième pic correspondant au n-ième point du spectre ($n = h_l(i)$), alors une valeur plus précise de la fréquence de ce pic $\hat{f}_{l,i} = \hat{n}_{l,i} F_e/N$ ($\hat{n}_{l,i}$ n'étant plus entier) et de son amplitude $\hat{A}_{l,i}$ peut être obtenue par une interpolation parabolique :

$$\hat{n}_{l,i} = n + \frac{1}{2}\frac{A_1 - A_{-1}}{2A_0 - A_{-1} - A_1}$$

et :

$$\hat{A}_{l,i} = A_0 + \frac{1}{4}(A_1 - A_{-1})(\hat{n}_{l,i} - n)$$

où, dans cette formule, $A_0 = |\widetilde{s_l}(n)|$, $A_{\pm 1} = |\widetilde{s_l}(n \pm 1)|$. Il faut noter que la précision est meilleure si l'interpolation de la fréquence et de l'amplitude est effectuée sur le spectre en dB [SER 89] (il suffit de remplacer A_0 et $A_{\pm 1}$ par $20 \log_{10} |\widetilde{s_l}(n)|$ et $20 \log_{10} |\widetilde{s_l}(n \pm 1)|$, puis de repasser le résultat en linéaire). Pour obtenir la phase interpolée, il faut effectuer l'interpolation de $\hat{n}_{\,l,i}$ comme précédemment, remplacer uniquement dans la formule donnant $\hat{A}_{l,i}$ les amplitudes A_0 et $A_{\pm 1}$ par les amplitudes complexes $\widetilde{s_l}(n)$ et $\widetilde{s_l}(n \pm 1)$ et prendre l'argument du résultat. Notons que la phase ne peut être estimée qu'à un multiple de 2π près.

Enfin, pour que l'amplitude estimée d'un pic corresponde à son amplitude effective dans le signal, il faut normaliser (diviser) l'amplitude précédente par le poids de la fenêtre sur chaque pic, $\sum_{k=0}^{M-1} w(k)/2$.

1.4.4. *Suivi de partiels*

Déterminer les pics du signal à chaque trame en se fondant uniquement sur les données de cette trame conduit à des listes indépendantes de pics non connectés. Or, un partiel naît avec la note, vit en fonction des variations de la note et meurt avant ou à la fin de la note. Sa durée de vie s'étale donc sur plusieurs trames. Il importe de relier les détections de pics effectuées sur chaque trame pour créer des « lignes » (*tracks* en anglais) de fréquence/amplitude traduisant l'évolution du partiel dans le temps. Dans la suite de ce paragraphe, la notion de partiel correspond donc à une suite de pics spectraux connectés. Le suivi est notamment indispensable pour effectuer une modification cohérente ou simplement une resynthèse.

Bien entendu, la technique qui consiste à associer les pics de deux trames successives ayant le même numéro ne peut pas fonctionner à cause de la présence de pics parasites et de la naissance ou de la mort de partiels de fréquence intermédiaire.

La technique décrite dans [MCA 86] consiste à déterminer de proche en proche les partiels par « continuité » : un partiel déterminé à la trame l est prolongé à la trame $l + 1$ par le pic dont la fréquence est la plus proche (à condition qu'elle appartienne à un intervalle centré sur la fréquence du partiel). Si un tel pic n'existe pas, le partiel « meurt ». De plus, un même pic ne peut appartenir qu'à un seul partiel, ce qui nécessite une technique de résolution de conflit lorsqu'un pic est le plus proche de deux partiels différents. Une fois tous les partiels traités, les pics restants sont considérés comme des partiels naissants. Lors de la naissance (respectivement de la mort) d'un partiel, il faut interpoler l'amplitude à zéro sur la trame précédente (respectivement suivante).

L'algorithme de McAulay et Quatieri est le suivant :

Pour $l \leftarrow 0$ à $L-2$ (pour chaque trame)
- Les fréquences $(f_{l,i})_{i=1,I_l}$ sont supposées triées par ordre croissant.
- Déclarer tous les partiels des trames l et $l+1$ comme non appariés
- Pour $i \leftarrow 1$ à I_l (pour chaque partiel de la trame l)
  - Trouver les pics non appariés de la trame $l+1$
  - dont la fréquence est dans l'intervalle $[f_{l,i}-\Delta, f_{l,i}+\Delta]$
  - Si aucun pic ne correspond
    - $f_{l,i}$ est déclaré mort et marqué comme apparié
  - Sinon
    - Soit $f_{l+1,j}$ la fréquence du pic le plus proche
    - Si le partiel non apparié de la trame l le plus proche de $f_{l+1,j}$ est $f_{l,i}$
      - Apparier $f_{l,i}$ et $f_{l+1,j}$: $suiv_l(i) = j$
      - Marquer $f_{l,i}$ et $f_{l+1,j}$ comme appariés
    - Sinon
      - Si le pic non apparié de la trame $l+1$ de fréquence $f_{l+1,j'}$
      - juste inférieure à $f_{l+1,j}$ est dans l'intervalle $[f_{l,i}-\Delta, f_{l,i}+\Delta]$
        - Apparier $f_{l,i}$ et $f_{l+1,j'}$: $suiv_l(i) = j'$
        - Marquer $f_{l,i}$ et $f_{l+1,j'}$ comme appariés
      - Sinon
        - $f_{l,i}$ est déclaré mort et marqué comme apparié
- Pour tous les pics j non appariés de la trame $l+1$
  - $f_{l+1,j}$ est déclaré naissant

Des améliorations ont été proposées à cet algorithme pour tenir compte d'effets à plus long terme que deux trames successives. En effet, il apparaît souvent qu'un partiel meurt et qu'un autre renaît quelques trames plus loin à la même fréquence, ce qui provient du fait que l'amplitude du partiel passe temporairement sous le seuil de rejet. Pour corriger cet inconvénient, Serra et Smith [SER 90] proposent de définir un état de « veille », retardant la mort des partiels inactifs et permettant leur réactivation s'ils peuvent s'apparier avec un pic avant un nombre déterminé de trames. Cependant, l'amplitude des partiels en état de veille étant nulle, ceci peut s'entendre comme un artefact lors de la resynthèse. Fitz et Haken [FIT 96] proposent de corriger ceci en intégrant les procédures de sélection dans celles de suivi de partiels pour créer un hystérésis dans le seuil de sélection : un partiel peut naître s'il est au-dessus du seuil de montée, mais ne peut mourir que s'il passe sous le seuil de descente qui est strictement inférieur au seuil de montée.

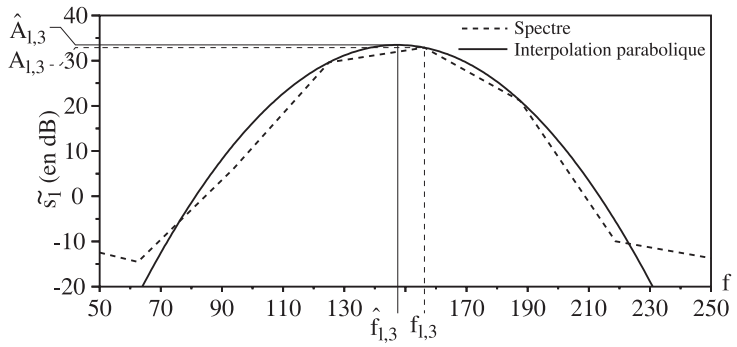


Figure 1.6. *Interpolation parabolique. En agrandissant le pic numéro 3 du spectre de la figure 1.5, on voit apparaître la discrétisation du spectre (en pointillé). La précision sur la fréquence du maximum local, $f_{l,3}$, peut être améliorée par une interpolation parabolique (en trait plein) fournissant la fréquence $\hat{f}_{l,3}$. L'interpolation a été effectuée sur les amplitudes en décibels.*

Enfin, d'autres critères sont utilisés tels qu'un nombre minimal de trames pour qu'un partiel soit valide ou qu'un nombre maximal autorisé de partiels simultanés. De même, lorsque le signal est supposé pseudo-périodique, l'algorithme se simplifie car il suffit alors de déterminer un partiel par pseudo-harmonique.

Le problème de suivi peut être aussi résolu par alignement temporel dynamique : apparier deux vecteurs de fréquences ordonnées revient à minimiser un critère de coût global de l'appariement, ce coût étant égal à la somme des écarts en fréquence de tous les partiels appariés. Cependant, l'effort pour formaliser ce problème peut être poussé un peu plus loin en tenant compte aussi de l'hypothèse de régularité de l'évolution des partiels. Ainsi, une technique de suivi par modèles de Markov cachés est décrite dans [DEP 93].

1.4.5. *Estimation de la partie bruit*

Nous avons vu, lors de la sélection des pics spectraux, des procédures destinées à éliminer les pics « parasites ». Il faut cependant se garder de penser que ce « bruit » n'est qu'une information parasite. En effet, les cas sont nombreux où ce « bruit » est une information primordiale du signal musical : le souffle d'un son de flûte, les transitoires d'attaque des tuyaux d'orgue, l'attaque des sons percussifs, le bruit des marteaux du piano, le frottement de l'archet du violon, les bruits d'aspiration, d'explosion et de frication dans la voix, etc.

La plupart des méthodes utilisées pour estimer la partie « bruit » du signal musical reposent sur un modèle de bruit additif : $s = x + b$, où x est un signal déterministe

(par exemple, pseudo-périodique) et b un signal aléatoire. Ce dernier signal est décrit par sa densité spectrale de puissance et il est souvent modélisé par un bruit blanc filtré. L'estimation de b consiste donc à estimer sa densité spectrale de puissance, soit en la décrivant explicitement en fonction de la fréquence, soit en la résumant dans le jeu de coefficients du filtre. Il faut noter que la phase ne joue aucun rôle et que l'estimation peut donc être réalisée sur le module du spectre.

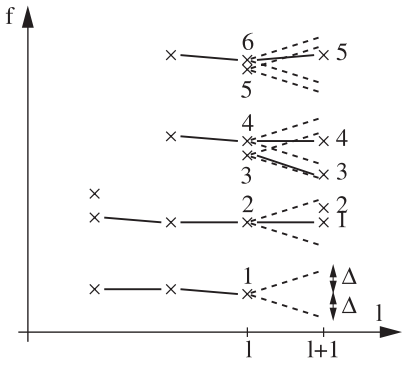


Figure 1.7. *Suivi de partiels. En déroulant l'algorithme sur les trames l et $l+1$, on obtient successivement les appariements suivants : $(l,1)$ meurt ; puis $(l,2)$ est apparié à $(l+1,1)$; ensuite $(l,3)$ est associé à $(l+1,4)$, mais comme ce dernier est plus proche de $(l,4)$, $(l,3)$ est apparié à $(l+1,3)$; puis $(l,4)$ est apparié à $(l+1,4)$; puis $(l,5)$ est associé à $(l+1,5)$, mais comme ce dernier est plus proche de $(l,6)$, $(l,5)$ meurt et c'est $(l,6)$ qui est apparié à $(l+1,5)$; finalement, $(l+1,2)$ est déclaré naissant.*

Néanmoins, en pratique, l'estimation du « bruit » n'est qu'un sous-produit de l'estimation de la partie « déterministe », obtenu par soustraction. La première étape est donc de calculer le signal de synthèse à partir des partiels pour pouvoir le retrancher au signal d'origine. Voici la formule de resynthèse décrite dans [MCA 86] : ayant estimé les amplitudes $\hat{A}_{l,i}$, fréquences $\hat{f}_{l,i}$ (les fréquences relatives sont donc $\hat{\nu}_{l,i} = \hat{f}_{l,i}/F_e$) et phases $\hat{\theta}_{l,i}$ de chaque partiel i pour chaque trame l, le signal de synthèse s'écrit comme une somme de sinusoïdes : $y_l(k) = \sum_{i=1}^{I_l} \hat{A}_{l,i}(k)\cos(\hat{\theta}_{l,i}(k))$ pour $k = 0, \ldots, D-1$ (les trames de synthèse sont donc de taille D et sont juxtaposées), où les amplitudes sont interpolées linéairement et les phases par un polynôme de degré 3 (assurant la continuité de la phase et de sa dérivée) :

$$\hat{A}_{l,i}(k) = \hat{A}_{l,i} + \frac{\hat{A}_{l+1,i} - \hat{A}_{l,i}}{D}k \qquad \text{pour } k = 0, \ldots, D-1$$

$$\hat{\theta}_{l,i}(k) = \hat{\theta}_{l,i} + 2\pi\hat{\nu}_{l,i}k + \left(\frac{\pi}{D}(\hat{\nu}_{l+1,i} - \hat{\nu}_{l,i}) + \frac{6\pi}{D^2}(\hat{M} - M_{opt})\right)k^2 - \frac{4\pi}{D^3}(\hat{M} - M_{opt})k^3$$

où la phase a été déroulée du facteur entier $\hat{M}$ (c'est-à-dire $\hat{\theta}_{l,i}(D) = \hat{\theta}_{l+1,i} + 2\pi\hat{M}$) le plus proche du réel $M_{\ opt}$ qui rend la courbe de phase la plus régulière [3] :

$$M_{opt} = \frac{1}{2\pi}(\hat{\theta}_{l,i} - \hat{\theta}_{l+1,i}) + \frac{D}{2}(\hat{\nu}_{l,i} + \hat{\nu}_{l+1,i})$$

La partie bruit est la différence entre ce signal synthétique reconstruit et le signal d'origine. Les spectres de bruit obtenus présentent alors des « trous » à la position des partiels. Or, ces « trous » étant répartis régulièrement, l'écoute d'un signal reconstruit à partir d'un tel spectre fait apparaître très clairement une hauteur (qui est celle du signal de départ). Cet artefact est supprimé en remplissant les « trous » laissés par les partiels. Plusieurs techniques sont utilisées selon les cas et la précision souhaitée de l'estimation. Serra et Smith [SER 90] utilisent l'interpolation linéaire pour calculer une enveloppe spectrale simplifiée du bruit : ils considèrent Q points régulièrement espacés en fréquence et affectent à chaque point l'amplitude maximale dans une bande centrée en ce point. Ensuite, la reconstruction du spectre est effectuée par interpolation linéaire entre les amplitudes de ces points.

Une autre méthode, fondée sur le modèle de production source/filtre, aboutit à une résolution par LPC (voir section 1.6). L'enveloppe est alors la réponse en fréquence d'un filtre tout-pôle ajusté par minimisation d'une erreur quadratique.

Remarquons que les techniques précédentes ne fournissent pas une décomposition exacte du signal en somme d'une partie « déterministe » et d'une partie « bruit », à cause de l'approximation effectuée sur la partie bruit. Dans certaines applications où la décomposition doit être exacte, il faut que la partie du bruit reconstruite à la position des partiels soit en quelque sorte ponctionnée sur ces partiels. C'est ce que proposent d'Alessandro *et al*. et Yegnanarayana *et al*. [DAL 98, YEG 98] dans un algorithme itératif de décomposition en parties périodique et apériodique.

1.5. Analyse du fondamental

En musique, la notion de fréquence fondamentale est liée à celle de hauteur mélodique. Sa connaissance est donc utile dans de nombreux cas, pour l'étude des échelles musicales en musicologie, du vibrato instrumental ou vocal, le suivi d'instruments, la transcription automatique de partitions, mais aussi comme information nécessaire à d'autres analyses comme l'analyse/synthèse synchrone au fondamental, certaines analyses de la qualité vocale, la décomposition périodique/apériodique, etc.

Les besoins des utilisateurs d'un système d'analyse du fondamental sont donc variés, allant de l'étiquetage du signal période par période à la détermination de la

3. Au sens du minimum du critère $\int_0^{DT_e} (\frac{d^2}{dt^2}\theta(t))^2\, dt$.

mélodie en termes de notes, en passant par l'estimation de la fréquence fondamentale à des instants régulièrement répartis.

Un grand nombre de techniques ont été développées à l'origine pour la parole [HES 83], donc pour un signal monophonique, variant sans cesse et à l'étendue relativement restreinte (de l'ordre de deux octaves). Les adapter au signal musical est possible à condition qu'elles puissent gérer de grandes étendues de fréquences fondamentales (jusqu'à six ou sept octaves). Cependant, la plus grosse difficulté est la polyphonie : lorsque plusieurs musiciens jouent ensemble ou lorsqu'il s'agit d'un instrument polyphonique, le son produit est beaucoup trop complexe pour qu'un système puisse, en toute généralité, en extraire les différentes notes. Les techniques les plus avancées permettent de séparer deux notes simultanées dans les bons cas [CHA 85, CHE 99, MAH 89, MAH 90].

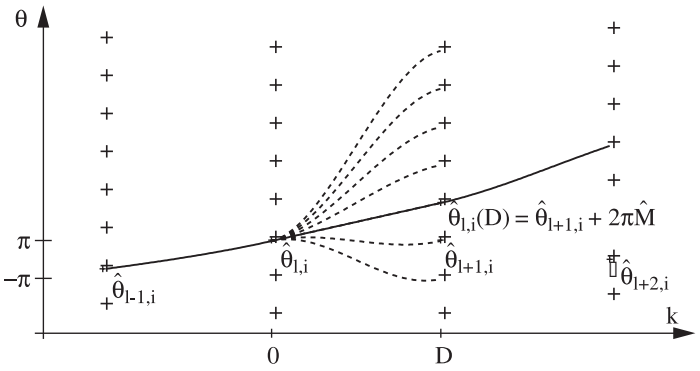


Figure 1.8. *Interpolation cubique de la phase. La phase interpolée est en trait plein. Puisque la phase est définie à un multiple de 2π près, la phase interpolée est la courbe la plus régulière passant par les croix. Par exemple, entre la trame l et la trame $l+1$, les courbes en pointillés représentent différentes possibilités d'interpolation à $2\pi M$ près, pour $M = -1, \ldots, 5$, l'optimum étant ici atteint pour $\hat{M} = 1$.*

1.5.1. *Fréquence fondamentale*

La notion de fréquence fondamentale est liée à la notion de périodicité. C'est donc une propriété du signal, à ne pas confondre avec la notion de « hauteur » qui en est le corrélat perceptif.

Un signal $s(t)$ est périodique s'il existe $T > 0$ tel que $s(t+T) = s(t)$, $\forall t$. Dans le cas où $s(t)$ n'est pas constant, il existe une plus petite valeur T_0 vérifiant cette propriété. La valeur T_0 est appelée période fondamentale et $F_0 = 1/T_0$ fréquence fondamentale ou « fondamental » tout court. Cependant, l'ensemble des valeurs vérifiant

la propriété de périodicité est l'ensemble des multiples de T_0 : $\forall k$, kT_0 est une période du signal (F_0/k est parfois appelée sous-harmonique). La période fondamentale est donc la plus petite des périodes.

C'est de cette définition que viennent toutes les difficultés liées à l'estimation du fondamental. En effet, une variation infime peut changer la période fondamentale en son double (ou en tout multiple k) : il suffit de modifier une valeur toutes les deux (ou k) périodes. Il en résulte une ambiguïté essentielle, se traduisant par des erreurs d'octave lors de la détermination du fondamental.

De plus, les signaux réels ne sont jamais strictement périodiques, donc la détermination du fondamental ne peut pas être fondée uniquement sur cette définition. Les source de variations sont nombreuses : variations de la fréquence fondamentale elle-même (vibrato, glissando), d'amplitude et tout autre paramètre de production (par exemple pour la voix, le conduit vocal et la qualité de la source sont très variables). Il en résulte l'impossibilité de définir le fondamental de façon unique.

Les algorithmes vont donc tenter d'élargir la notion de périodicité stricte, en exploitant les notions proches : la ressemblance entre signaux, l'harmonicité, etc. Cependant, il est très courant que les idées développées bouclent : elles nécessitent la connaissance de la fréquence fondamentale pour l'estimer...

Une dernière remarque est que si l'on peut faire l'hypothèse que le fondamental est situé dans un intervalle d'octave, n'importe quelle technique élémentaire parviendra à l'estimer.

1.5.2. *Concevoir un algorithme de détermination du fondamental*

Le schéma général se décompose en trois étapes : prétraitement, traitement principal et post-traitement. Le prétraitement consiste à mettre en forme le signal acoustique, c'est-à-dire le numériser et lui appliquer certaines transformations globales de type filtrage (passe-bas, filtrage inverse, *clipping*, etc.). Puis, dans le cas de méthodes à court terme, une représentation de ce signal est obtenue par application d'une transformation : représentation temporelle à court terme (extraction de trames successives avec recouvrement et fenêtrage éventuels), représentation fréquencielle ou cepstrale à court terme (transformation temps-fréquence). Le traitement principal peut être vu en général comme le calcul d'une fonction de périodicité $FP(f_0)$ utilisant les informations fournies par le prétraitement, suivi d'une décision sur le meilleur candidat. La fonction de périodicité mesure pour chaque fondamental candidat f_0 à quel point il « explique » la périodicité du signal. La décision devrait se résumer à la détermination d'un maximum : $\widehat{f_0} = \max_{f_0} FP(f_0)$. Cependant, la plupart des méthodes conservent l'ambiguïté d'octave entre les différents candidats. Par exemple, l'autocorrélation d'un signal périodique est périodique de même période. Ceci conduit presque toujours à

corriger *a posteriori* les résultats de $FP(f_0)$ ou à prendre une décision tenant compte à la fois de $FP(f_0)$ et de f_0. Enfin, le post-traitement sert à rendre les courbes de f_0 plus lisses grâce à des techniques de lissage (par exemple filtrage médian) ou de suivi (programmation dynamique ou modèles de Markov cachés).

Quel que soit l'algorithme choisi pour déterminer le fondamental, un certain nombre de paramètres sont déterminants. Ce sont :

1) $F_{0\min}$: la plus petite fréquence fondamentale explorée. C'est le paramètre le plus critique. Il influence le taux d'erreurs grossières par sous-octaviation et doit donc être choisi « au plus juste » ;

2) $F_{0\max}$: la plus grande fréquence fondamentale explorée ;

3) $F_{\max}$: la plus haute fréquence utile du signal. Le nombre maximal d'harmoniques prises en considération dépendra donc du fondamental et sera égal à $F_{\max}/F_{0\min}$ pour $F_0 = F_{0\min}$ et à $F_{\max}/F_{0\max}$ pour $F_0 = F_{0\max}$.

Pour évaluer les tendances d'un algorithme à l'octaviation, il suffit de considérer un signal parfaitement périodique de fondamental F_0 et d'examiner les réponses de l'algorithme aux fréquences multiples et sous-multiples de F_0. La grande majorité des algorithmes donnera, à la sortie du traitement principal, une réponse identique (ou presque) pour le vrai fondamental et pour ses sous-multiples. Les algorithmes qui traitent ce problème dans leur principe sont parmi les plus robustes.

1.5.3. *Prétraitements*

Le but du prétraitement est de mettre en évidence la structure périodique du signal par l'intermédiaire d'une représentation bien choisie, en supprimant autant que possible l'information « inutile » et en renforçant (ou en extrayant) l'information « utile ».

La première idée qui vient à l'esprit est donc l'utilisation d'une procédure de décomposition du signal en parties périodique et apériodique qui permettrait de se concentrer uniquement sur la partie périodique. Malheureusement, ces procédures nécessitent en général l'information du fondamental pour fonctionner et ne sont donc pas utilisables pour la déterminer.

En fait, l'essentiel de l'information concernant le fondamental étant contenu dans les basses fréquences (ou du moins dans les premières harmoniques), le prétraitement le plus utilisé est un filtrage passe-bas souvent associé à un sous-échantillonnage. Ceci se justifie par le fait que l'amplitude des harmoniques suit une tendance décroissante avec le numéro d'harmonique et que, parallèlement, les composantes de bruit ont une répartition spectrale plutôt située du médium à l'aigu. La fréquence de coupure du filtre passe-bas est choisie en fonction de la plus haute fréquence fondamentale du

signal analysé (elle doit au moins lui être supérieure) et du nombre d'harmoniques « utiles » (une heuristique consiste à prendre $F_{\max} = 3$ à 5 fois $F_{0\max}$).

Pour certains algorithmes fonctionnant sur la représentation temporelle du signal, il peut être très utile de supprimer la « composante continue ». Pour ce faire, il suffit d'appliquer sur le signal un filtrage passe-haut avec une faible fréquence de coupure (inférieure à $F_{0\min}$).

Dans certains cas, le modèle de production source/filtre peut s'appliquer. Ce modèle indique que le signal sonore est produit par filtrage (le résonateur) d'un signal de source (l'excitation) ; dans ce cas, l'information relative au fondamental est portée par le signal de source. Il est donc plus intéressant d'estimer le fondamental sur le signal de source, dont une estimation peut être obtenue à partir du signal sonore par des procédures de « filtrage inverse ». Une telle procédure est décrite au paragraphe 1.6.2. Le modèle de production source/filtre est très bien adapté aux signaux de parole et la grande majorité des algorithmes de détermination du fondamental utilise un filtrage inverse comme prétraitement. Il convient aussi à la voix chantée tant que le fondamental n'est pas trop aigu. Pour les signaux instrumentaux, ce prétraitement ne présente d'intérêt que si la source est riche en harmoniques. Le choix de l'ordre du modèle est critique : pour la voix, il peut être choisi égal à deux fois la fréquence maximale exprimée en kHz plus 2 (par exemple, si la fréquence d'échantillonnage est de 16 kHz, choisir un ordre de 18) [4].

Enfin, pour éliminer l'influence des variations d'amplitude sur le signal, il est possible d'effectuer un contrôle automatique de gain, par exemple en multipliant le signal par l'inverse de son enveloppe temporelle (voir section 1.3).

Bien entendu, ces différents prétraitements peuvent être utilisés simultanément.

1.5.4. *Méthodes temporelles*

Puisque la fréquence fondamentale d'un son périodique est celle de son harmonique 1, la première idée qui vient à l'esprit est d'extraire la fréquence de cette harmonique, par exemple par simple filtrage passe-bas. Bien entendu, le choix de la fréquence de coupure est critique et nécessite, pour être optimal, de connaître approximativement la fréquence fondamentale...

Une idée très répandue est de compter directement sur le signal des événements qui se répètent comme les passages par zéro ou par un (ou deux) seuils adaptatifs (en comptant uniquement les fronts montants pour éliminer les effets de composante

4. Cela correspond à associer une paire de pôles de la fonction de transfert à chaque formant, en supposant qu'il y a un formant tous les 1 000 Hz.

continue). Cela ne fonctionne que si le signal analysé est presque sinusoïdal, par exemple s'il ne contient essentiellement que l'harmonique 1. L'idée développée par Kuhn [KUH 90] est de décomposer le signal sonore en le passant dans un banc de filtres en octave (après un contrôle automatique de gain). Sur chaque sortie de filtre est mesurée l'amplitude $A(i)$ et la période $T(i)$ par un simple comptage des passages par zéro. Il reste alors à prendre la décision définitive du fondamental en fonction du résultat des amplitudes et périodes indiquées par chaque filtre.

L'algorithme de décision est le suivant :

$A_{\max} \leftarrow \max_i\big(A(i)\big)$
Si $A_{\max} < seuil\_silence$
  $T_0 \leftarrow -1$
Sinon
  $seuil \leftarrow A_{\max}/4$
  $T_0 \leftarrow -1$
  $convient \leftarrow FAUX$
  $filtre \leftarrow 0$
  Tant que $filtre < n$ et $convient = FAUX$
    Si $A(filtre) > seuil$
      Si $T(filtre)$ est raisonnable pour $filtre$
        $T_0 \leftarrow T(filtre)$
      Sinon si $filtre < n-1$ et $T(filtre+1)$ raisonnable pour $filtre+1$
        $T_0 \leftarrow T(filtre+1)$
      Sinon
        $T_0 \leftarrow -1$
      $convient \leftarrow VRAI$
    Sinon
      $filtre \leftarrow filtre + 1$

Cet algorithme révèle clairement toute l'ambiguïté de la définition de la période fondamentale : il choisit la plus petite période (la première ou éventuellement la deuxième) dont l'amplitude est parmi les plus grandes.

Certaines méthodes analysent des points remarquables du signal (minimums, maximums, passages par zéro) [COO 96, GOL 69].

L'inconvénient majeur des méthodes temporelles est leur mauvaise robustesse aux bruits. Leur avantage est d'être très rapides.

1.5.5. *Méthodes temporelles à court terme*

L'idée sous-jacente des méthodes temporelles à court terme est d'exploiter la ressemblance entre deux tranches de signal séparées par une période. La période estimée sera définie par l'écart de temps entre ces deux tranches. Le prétraitement consiste donc au moins à faire une analyse à court terme du signal, produisant les trames s_l sur lesquelles sera estimée une valeur de fondamental par trame.

Une mesure courante de ressemblance est l'autocorrélation du signal. L'algorithme le plus simple estime le fondamental comme le maximum de la fonction de périodicité suivante :

$$FP_{Autoc,l}(\tau) = \frac{1}{Norme} \sum_{k=0}^{M-1-\tau} s_l(k) s_l(k+\tau)$$

et :

$$f_0 = F_e/\tau$$

où la normalisation par $Norme = (\sum_{k=0}^{M-1-\tau} s_l^2(k) \sum_{k=0}^{M-1-\tau} s_l^2(k+\tau))^{1/2}$ permet de comparer la fonction de périodicité à un seuil absolu pour décider si la trame correspond à un signal périodique ou non. Comme nous l'avons déjà mentionné, cette fonction de périodicité étant périodique de même période que le signal, les deux approches – qui sont d'extraire simplement le maximum ou de choisir la période du premier maximum local de cette fonction – donnent rarement le bon résultat. La plupart des implémentations de cet algorithme proposent soit un mélange de ces deux approches, soit de pondérer la fonction de périodicité (par une exponentielle décroissante, par exemple) pour défavoriser les multiples de la période fondamentale.

Une approche un peu différente et qui donne en général de meilleurs résultats est de choisir le minimum de l'AMDF ou *average magnitude difference function*, qui consiste à faire la différence entre une trame du signal et une version décalée de cette trame :

$$FP_{AMDF,l}(\tau) = \frac{1}{Norme} \sum_{k=0}^{M-1} |s_l(k) - s_l(k+\tau)|$$

et :

$$f_0 = F_e/\tau$$

avec par exemple $Norme = \sum_{k=0}^{M-1} |s_l(k)|$. En théorie, il y aura un minimum à tous les multiples de la période fondamentale. Mais lorsque le signal n'est pas tout à fait périodique, plus le décalage est grand, moins la trame décalée ressemblera à la trame

de départ et plus l'AMDF sera grande. Ainsi, l'algorithme défavorise de façon implicite les périodes multiples. Cependant, des heuristiques sont presque toujours utilisées pour défavoriser les grandes périodes [CHE 91].

Une extension intéressante des fonctions précédentes consiste à calculer la ressemblance sur une fenêtre de taille variable, correspondant par exemple à la période candidate. C'est ce que propose l'algorithme SRPD (*super resolution pitch determination*) [MED 91], fondé sur une intercorrélation entre deux fenêtres successives de taille égale à la période candidate, normalisée pour tenir compte de la variation du nombre de points :

$$FP_{SRPD,l}(\tau) = \frac{\sum_{k=0}^{\tau-1} s_l(k)s_l(k+\tau)}{\left(\sum_{k=0}^{\tau-1} s_l^2(k)\sum_{k=0}^{\tau-1} s_l^2(k+\tau)\right)^{\frac{1}{2}}}$$

et :

$$f_0 = F_e/\tau$$

Quelle que soit la méthode choisie, un des problèmes liés à la discrétisation est la précision de la mesure : la période est mesurée par un nombre de points de décalage, donc la précision est limitée par la période d'échantillonnage du signal. Par exemple, à $F_e = 16\,000$ Hz sur un Do aigu ($F_0 = 1\,046$ Hz), la précision sur la période étant d'un échantillon, soit $0{,}0625$ ms, la fréquence sera déterminée au mieux à ± 33 Hz, donc à un demi-ton près. Une solution est d'interpoler la fonction de périodicité au voisinage du maximum (/minimum) correspondant à la période estimée [GEO 96, MED 91] ou d'utiliser plusieurs fenêtres successives [BRO 89].

1.5.6. *Méthodes fréquencielles à court terme*

Fréquenciellement, la périodicité se traduit par le fait que les partiels du signal sont harmoniques. Un grand nombre de méthodes exploitent cette information en travaillant sur le spectre du signal.

Le principe de ces méthodes est donc d'effectuer une analyse spectrale à court terme (TFCT, voir paragraphe 1.2.3) fournissant un spectre à court terme $\widetilde{s}_l$ par trame $l = 0, \ldots, L-1$ et, pour chaque trame, de calculer une fonction de périodicité $FP(f_0)$ sur $\widetilde{s}_l$ et d'en déduire l'estimation d'une fréquence fondamentale (par exemple par l'indice du maximum de $FP(f_0)$).

Le spectre peut être exploité directement ou n'être qu'une étape pour l'estimation des partiels (voir section 1.4). Un certain nombre de prétraitements du signal et du spectre sont souvent utilisés pour renforcer, isoler ou égaliser les harmoniques.

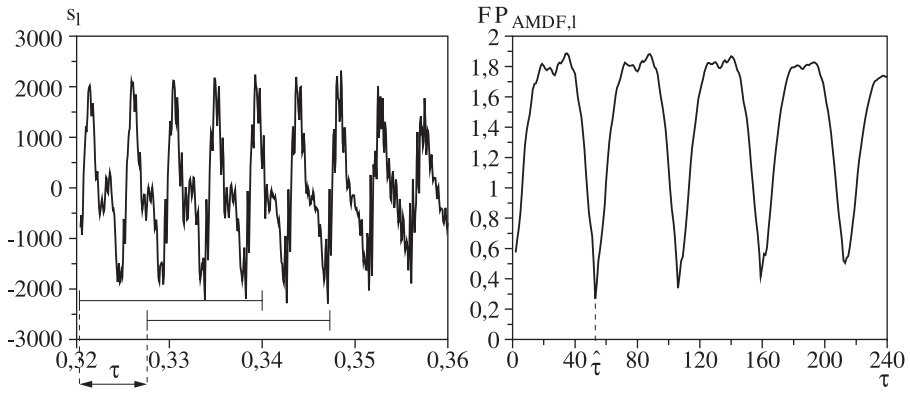


Figure 1.9. *Estimation de la fréquence fondamentale par AMDF. Une trame de signal (à gauche) est comparée avec une version décalée de τ pour fournir un point de la fonction de périodicité (à droite). Le décalage qui provoque la plus petite valeur d'AMDF est considéré comme la période du signal. Ici, il s'agit d'un signal de parole à 12 kHz, analysé pour des périodes allant de 30 à 400 points (correspondant à des f_0 de 50 Hz à 400 Hz). La période estimée est ici de $\hat{\tau} = 53$, soit $\widehat{f_0} = 226\,Hz$ +/− $4\,Hz$. Remarquez la présence de minimums locaux importants à tous les multiples de $\hat{\tau}$.*

Le principe de la compression spectrale consiste à faire la somme du spectre et de toutes les versions « compressées » d'un facteur entier de ce spectre. Ainsi, toutes les contributions du spectre aux positions des harmoniques se retrouvent sommées à la position du fondamental.

Quant à l'appariement d'harmoniques, son principe est d'apparier les partiels du signal à des valeurs de référence choisies pour être les harmoniques d'un candidat au fondamental. Le spectre de référence a donc la forme d'un peigne. Une mesure pour chaque candidat f_0 est alors déduite de l'appariement effectué pour ce f_0.

L'idée la plus simple consiste, pour un candidat au fondamental f_0 donné, à faire l'intercorrélation entre le spectre et le peigne de fondamental f_0 :

$$FP_l(f_0) = \int_f \perp\!\!\!\perp_{f_0}(f)|\widetilde{s_l}(f)|\,df = \sum_i |\widetilde{s_l}(if_0)|$$

Notons que la somme est souvent remplacée par un produit, ce qui revient à calculer la somme sur le spectre exprimé en dB.

Il est remarquable que l'intercorrélation avec un peigne est équivalente à la somme des spectres compressés. Nous ne développerons que la première.

L'algorithme présenté par Martin [MART 81, MART 82] est fondé sur ce principe d'intercorrélation avec une fonction peigne avec quelques améliorations notables. Tout d'abord, le spectre étant discret, la fonction de périodicité est calculée sur les points du spectre d'indice n_0 entier :

$$FP_{IFP,l}(n_0) = \sum_{i=1}^{I(n_0)} \alpha_i |\widetilde{s_l}(in_0)|$$

le fondamental candidat étant $f_0 = n_0 F_e/N$. Ensuite, comme l'indique la formule précédente, les dents du peigne sont pondérées par les coefficients rapidement décroissants $\alpha_i = i^\gamma$, $\gamma < 0$. Ce point est critique pour la robustesse de l'algorithme. Une valeur de $\gamma = -0{,}5$ convient pour la plupart des applications.

Contrairement à ce que l'on pourrait penser, la décroissance des dents du peigne n'a rien à voir avec la tendance naturelle de décroissance des spectres des signaux musicaux. En fait, si les spectres avaient des amplitudes croissantes, il faudrait quand même prendre un peigne décroissant. La décroissance du peigne permet de défavoriser les sous-multiples du vrai fondamental : le candidat de fréquence moitié du vrai fondamental sommera toutes les harmoniques, mais avec les coefficients α_{2i} qui sont inférieurs aux α_i.

De plus, dans l'algorithme de Martin, seuls les partiels importants du signal sont pris en considération dans le calcul. Pour cela, la méthode proposée est, plutôt que de prendre tous les points du spectre, d'effectuer une sélection des pics du spectre candidats aux harmoniques (voir paragraphe 1.4.2) et de reconstruire un spectre en n'utilisant que la fréquence et l'amplitude de ces pics. Pour tenir compte de l'inharmonicité éventuelle, les pics sont reconstruits avec une forme de parabole. Dans la formule précédente, $\widetilde{s_l}$ doit donc être remplacé par le spectre reconstruit.

Enfin, la précision sur la détermination de la fréquence fondamentale est limitée par l'intervalle de fréquence entre deux points du spectre. Une méthode consiste à suréchantillonner le spectre reconstruit pour obtenir la précision voulue, par exemple en utilisant des splines [HER 88]. De plus, l'appariement d'harmoniques (en associant par exemple l'harmonique k avec le pic le plus proche de $k\,\widehat{f_0}$) permet de réestimer le fondamental par régression sur les fréquences des harmoniques pondérées par leurs amplitudes :

$$\widehat{f}_{0\text{précis}} = \frac{\sum_k \hat{A}_{l,i(k)} \hat{f}_{l,i(k)}/k}{\sum_k \hat{A}_{l,i(k)}}$$

où $i(k)$ désigne le numéro du pic apparié à l'harmonique k.

De nombreuses autres méthodes entrant dans la même catégorie ont été développées, essentiellement pour la parole. Paliwal et Rao [PAL 83] proposent de mesurer la distance entre les partiels du spectre et un peigne dont les amplitudes sont

situées sur une enveloppe spectrale estimée sur ce spectre. La méthode de Duifhuis *et al*. [DUI 82] est fondée sur la théorie perceptive de Goldstein et sur la notion de crible harmonique : c'est une sorte de « tamis » qui rejette tout ce qui n'est pas harmonique à une précision près. Dans le cas de sons riches en harmoniques, le maximum du cepstre dans une zone de valeurs possibles correspond à la période fondamentale [NOL 67]. Il existe aussi des approches utilisant des connaissances plus spécifiques sur les sons étudiés : Doval [DOV 94] propose d'effectuer un apprentissage statistique des caractéristiques des harmoniques (nombre, amplitudes, inharmonicités, ce qui permet d'analyser des spectres à partiels inharmoniques) et du « bruit », Sieger et Tewfik [SIE 98] utilisent un dictionnaire d'harmoniques par fondamental.

1.5.7. *Suivi de la fréquence fondamentale*

Le besoin des utilisateurs étant de disposer d'une valeur de fondamental à chaque instant, la décision habituellement prise est locale et consiste à extraire le maximum de la fonction de périodicité. Les erreurs les plus courantes sont alors des erreurs grossières (d'octave) localisées, typiquement un aller-retour entre le fondamental et sa moitié ou parfois son double. Or, il est clair qu'aucun instrument ne pourra produire une telle séquence de fréquences fondamentales. La présence de telles erreurs est due au fait que l'information sur l'évolution du fondamental dans le temps n'est pas utilisée.

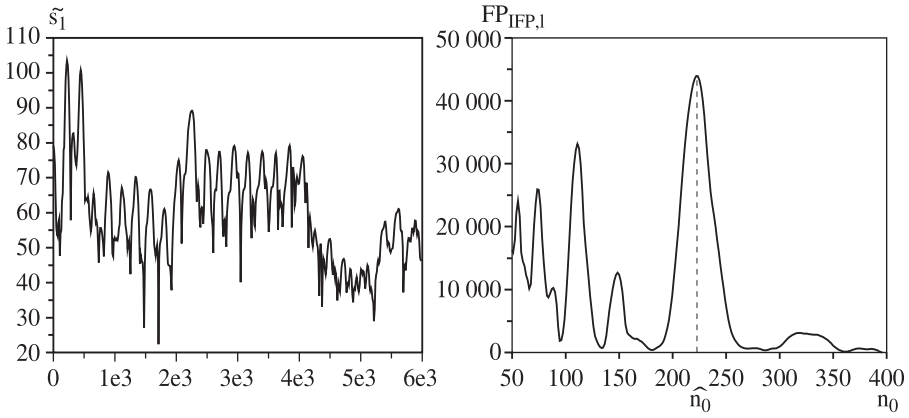


Figure 1.10. *Estimation de la fréquence fondamentale par intercorrélation avec une fonction peigne. Le spectre est corrélé avec un peigne dont les dents sont écartées de n_0 et le résultat fournit un point de la fonction de périodicité. La valeur fournissant la meilleure corrélation ($\widehat{n_0}$) correspond à la fréquence fondamentale. Les conditions sont identiques à celles de la figure 1.9 (même signal) et la fréquence estimée est $\widehat{f_0} = \widehat{n_0} F_e/N = 223$ +/− 1 Hz. Remarquez la présence de maximums locaux atténués aux sous-multiples de $\widehat{f_0}$.*

Le suivi de fréquence fondamentale est alors une technique permettant d'améliorer très nettement les résultats d'estimation. Une des solutions parmi les meilleures est le suivi par programmation dynamique ou, dans sa version mieux formalisée et présentée ici, par modèles de Markov cachés. Ils permettent de prendre une décision globale fournissant une séquence de valeurs du fondamental optimale au sens où elle tient compte de l'évolution possible du fondamental entre deux instants successifs tout en expliquant bien les caractéristiques observées à un instant donné. Une description du fonctionnement et des algorithmes des modèles de Markov cachés peut être trouvée dans [PAP 84, RAB 86].

La structure du modèle de Markov caché (MMC) est un ensemble d'états. Chaque état est associé à un fondamental possible (candidat). On parlera donc de l'état f_0. Tous les états du MMC sont connectés entre eux, ce qui signifie qu'il est *a priori* possible de passer d'un fondamental à n'importe quel autre entre deux instants successifs.

Le fonctionnement direct du MMC est d'émettre le signal observé (celui dont on cherche à connaître la séquence des fréquences fondamentales) trame par trame, de la façon suivante : à l'instant t_l, le MMC se trouve dans l'état f_0 et émet la trame de signal s_l (ou plus exactement sa représentation), puis il effectue une transition vers un autre état (ou vers le même état) et passe à l'instant suivant. Ce processus est itéré jusqu'à obtenir toutes les trames de signal.

Pour pouvoir fonctionner, le MMC a donc besoin, d'une part, de la densité de probabilité d'apparition des observations dans chaque état (la fonction de périodicité $FP_l(f_0)$ est considérée comme une mesure de cette densité) et, d'autre part, de la probabilité de transition d'un état dans un autre, notée $tr(f'_0, f_0)$, qui est justement l'information qui traduit l'évolution du fondamental d'un instant à l'autre. Cette probabilité pourra être apprise (apprentissage statistique paramétrique ou non) ou modélisée *a priori*.

Il apparaît donc que le MMC est un modèle d'émission d'un signal à partir d'une séquence de fréquences fondamentales. Or, le suivi est exactement le problème inverse : celui de retrouver la séquence des fréquences fondamentales connaissant le signal observé. En termes de Markov, le problème est de trouver la séquence optimale des états qui ont pu émettre le signal donné.

L'algorithme permettant de résoudre ce problème est l'algorithme de Viterbi dont nous présentons ici le détail. Considérons le tableau FP dont les abscisses représentent les instants t_l d'analyse des trames de signal, dont les ordonnées représentent les états possibles f_0. Remplissons ce tableau par colonne avec les probabilités d'émission de la trame du temps t_l par l'état f_0 (ce sont les valeurs des fonctions de périodicité de l'instant correspondant, $FP_l(f_0)$). Considérons les chemins qui passent par une et une seule case du tableau à chaque instant et définissons la probabilité d'un tel chemin par le produit de toutes les probabilités d'émission correspondant aux cases traversées

par le chemin et des probabilités de transition entre deux cases successives. Alors, l'algorithme trouve le chemin optimal au sens où il maximise sa probabilité.

Bien entendu, l'algorithme n'explore pas tous les chemins possibles mais construit de proche en proche le tableau noté $\boldsymbol{C}[l, f_0]$ des probabilités de tous les sous-chemins optimaux et choisit celui dont la probabilité finale est maximale. Enfin, pour pouvoir récupérer effectivement la séquence d'états de ce chemin, à chaque transition, l'état précédent est mémorisé dans $\mathbf{prec}[l, f_0]$. En pratique, il y a quatre étapes :

– initialisation : Pour $f_0 \leftarrow F_{0\min}$ à $F_{0\max}$, $\boldsymbol{C}[0, f_0] = FP_0(f_0)$

– propagation : Pour $l \leftarrow 1$ à $L-1$
  Pour $f_0 \leftarrow F_{0\min}$ à $F_{0\max}$
    $\boldsymbol{C}[l, f_0] = \max_{f'_0}\big(\boldsymbol{C}[l-1, f'_0] tr(f'_0, f_0)\big) FP_l(f_0)$
    $\mathbf{prec}[l, f_0] = \arg\max_{f'_0}\big(\boldsymbol{C}[l-1, f'_0] tr(f'_0, f_0)\big)$

– terminaison :
  $proba_{opt} = \max_{f_0}\big(\boldsymbol{C}[L-1, f_0]\big)$
  $\widehat{f_0}[L-1] = \arg\max_{f_0}\big(\boldsymbol{C}[L-1, f_0]\big)$

– recherche arrière : Pour $l \leftarrow L-2$ à 0, $\widehat{f_0}[l] = \mathbf{prec}\big[l+1, \widehat{f_0}[l+1]\big]$

La séquence optimale des valeurs de fréquence fondamentale est alors :

$$\widehat{\boldsymbol{f_0}} = \big(\widehat{f_0}[0], \widehat{f_0}[1], \ldots, \widehat{f_0}[L-1]\big)$$

Il est clair que dans son principe, cette méthode de suivi peut s'adapter à n'importe quel algorithme de détermination du fondamental. Cependant, en toute rigueur, il faut transformer au préalable la fonction de périodicité FP en densité de probabilité. La pratique la plus courante est de pondérer l'importance relative de FP et des probabilités de transition tr pour équilibrer les contraintes de régularité d'évolution et d'adéquation locale.

Un choix standard pour les probabilités tr est une gaussienne sur l'écart relatif des fréquences :

$$tr(f'_0, f_0) = \frac{e^{-0{,}5(\frac{f'_0 - f_0}{0{,}5(f'_0 + f_0)}/\sigma)^2}}{\sigma\sqrt{2\pi}}$$

S'il n'est pas prévu d'apprentissage, une heuristique consiste à choisir σ de l'ordre de T_a ($\sigma = 0{,}4T_a$ convient à l'utilisation directe de l'algorithme décrit au paragraphe 1.5.6).

Le désavantage de cet algorithme est qu'il ne fonctionne pas en temps réel. Il est toujours possible de l'utiliser par bloc, mais des versions sous-optimales en temps réel existent [RAP 99].

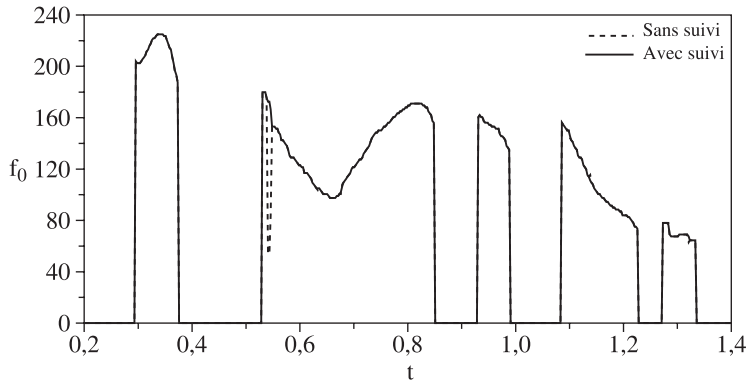


Figure 1.11. *Suivi de fréquence fondamentale. Cette phrase a été analysée avec l'algorithme de Martin qui a fourni une fonction de périodicité pour chaque trame, puis l'algorithme de suivi décrit dans le texte a été appliqué sur les fonctions de périodicité. Remarquez que les erreurs grossières locales sont corrigées (exemple à l'instant $t = 0{,}54\,s$). Sur cet exemple, les parties de signal où la fréquence fondamentale n'est pas définie ont été mises à zéro.*

1.6. Analyse de l'enveloppe spectrale

Pour la plupart des instruments, le son est produit par un excitateur, puis mis en forme par un résonateur. Le signal de l'excitateur (la source) est composé d'harmoniques dont l'amplitude est généralement régulièrement décroissante en fonction du numéro d'harmonique. De son côté, le résonateur agit comme un filtre dont la réponse en fréquence est une fonction continue de la fréquence présentant des maximums et des vallées qui vont « modeler » l'amplitude des harmoniques produites par la source. L'amplitude d'une harmonique est alors la somme (en dB) des amplitudes de la composante correspondante dans la source et de la réponse du filtre.

Le nombre d'harmoniques significatives d'un signal musical peut être parfois très grand et la fidélité de l'analyse nécessite de les estimer à court terme, ce qui produit un nombre énorme de paramètres à conserver pour d'éventuelles transformations ou même juste pour le stockage. L'enveloppe spectrale est alors un moyen économique de codage de l'amplitude des harmoniques en fonction de la fréquence par un jeu de coefficients en nombre très réduit (de l'ordre d'une vingtaine pour des signaux très complexes). Bien entendu, une enveloppe spectrale peut être calculée aussi bien sur des signaux de bruit que sur des signaux pseudo-périodiques.

Les utilisations de l'enveloppe spectrale en musique sont variées [LAN 89] : modification du fondamental d'une voix sans changer la qualité vocalique ni le débit, modification du débit, transformation de la source seule (en voix chuchotée ou

alors parfaitement tonale), synthèse croisée ou encore toute application nécessitant la mesure de distance de « timbre » [DEP 94].

1.6.1. *Enveloppe spectrale*

Une enveloppe spectrale paramétrique peut être vue comme une fonction appartenant à une famille de fonctions (spline, réponse en fréquence de filtres linéaires, cepstres, etc.) décrite par un jeu de paramètres. Une enveloppe est donc associée de façon biunivoque à un vecteur de paramètres.

Le principe général de l'estimation d'enveloppe est alors de déterminer le jeu de paramètres qui minimise un critère d'erreur mesurant l'écart entre l'enveloppe et les points du spectre. D'un point de vue plus pratique, il s'agit de faire passer une courbe régulière par ces points ou proche de ces points.

Un cas plus simple est celui des enveloppes linéaires par morceaux (non paramétriques), définies par un petit nombre de points spectraux et dont l'estimation est directe, puisqu'il suffit de trouver le maximum du spectre par bandes de fréquences [SER 90].

1.6.2. *Estimation par prédiction linéaire (LPC)*

Quand le modèle source/filtre s'applique bien au signal analysé, il est naturel d'estimer l'enveloppe spectrale par la réponse en fréquence de la partie filtre. Généralement, le modèle fait l'hypothèse sur la source qu'elle est composée d'impulsions pseudo-périodiques ou qu'il s'agit de bruit blanc. Son spectre est donc plat (c'est-à-dire que la pente spectrale est de 0 dB/oct). De plus, le filtre est supposé tout-pôle (autorégressif) et sa fonction de transfert est donc de la forme $H(z) = G/\sum_{i=0}^{p} a(i)z^{-i}$, où les $a(i)$ sont les coefficients du filtre, d'ordre p, et où $a(0) = 1$.

Cependant, les sources des instruments de musique ont plutôt des spectres décroissant en $1/f$ (la pente spectrale est de -6 dB/oct) ou même décroissant plus vite. Il est alors utile de « préaccentuer » le signal par une fonction qui ajoute une pente de $+6$ dB/oct pour qu'il satisfasse mieux aux hypothèses du modèle, par exemple par l'application d'un filtre dérivateur du type $x(k) = s(k) - \alpha s(k-1)$, où $\alpha < 1$, par exemple $\alpha = 0{,}976$.

Pour estimer les coefficients du filtre, la technique la plus utilisée, la prédiction linéaire (LPC en anglais), a été développée pour des applications de codage de la parole. C'est une technique des moindres carrés appliquée au critère d'erreur de prédiction des échantillons de signal : $err = \sum_{k=0}^{M-1} (s(k) - \hat{s}(k))^2$, où

$\hat{s}(k) = \sum_{i=1}^{p} a(i)s(k-i)$ est le signal de prédiction. Le lecteur trouvera une description détaillée des développements théoriques et algorithmiques dans [MAR 76]. Cependant, nous donnons ici un algorithme permettant de calculer les coefficients du filtre et son gain. Il s'agit de la méthode d'autocorrélation associée à l'algorithme de Durbin.

Considérons une trame de signal $s_l(k)$, $k = 0, \ldots, M-1$ issue d'une analyse à court terme (voir paragraphe 1.2.3) dont nous souhaitons estimer l'enveloppe spectrale. L'algorithme consiste à estimer les coefficients de corrélation $R(i)$ de cette trame, puis les coefficients du filtre $a(i)$ et son gain G :

– $R(i) = \sum_{k=0}^{M-i-1} s_l(k)s_l(k+i)$, $i = 0, \ldots, p$;

– algorithme de Durbin :

$err \leftarrow R(0)$
$a(0) \leftarrow 1$
Pour $i \leftarrow 1$ à p
  $a(i) \leftarrow \left(-\sum_{j=0}^{i-1} a(j)R(i-j)\right)/err$
  Pour $j \leftarrow 1$ à $i-1$
    $b(j) \leftarrow a(j) + a(i)a(i-j)$
  Pour $j \leftarrow 1$ à $i-1$
    $a(j) \leftarrow b(j)$
  $err \leftarrow \left(1 - a(i)^2\right) err$
Les $a(i)$ sont les coefficients du filtre et err est l'erreur de prédiction

– calcul du gain : $G = \sqrt{\sum_{i=0}^{p} a(i)R(i)}$.

Une fois estimé le filtre, il est simple d'estimer la source par « filtrage inverse » : $e(k) = \frac{1}{G}\sum_{i=0}^{p} a(i)s_l(k-i)$, $k = 0, \ldots, M-1$ en considérant les échantillons d'indice négatif comme nuls.

Si une préaccentuation a été utilisée avant la LPC, il faut maintenant désaccentuer le résultat du filtrage inverse pour obtenir le signal de source estimé.

Le tracé de l'enveloppe sur N points est simple : il s'agit du module de la réponse en fréquence du filtre. Le calcul peut se faire par TFD, car $ENV(n) = |H(z)|_{e^{j2\pi n/N}}| = G/|\sum_{i=0}^{p} a(i)e^{-j2\pi ni/N}|$ où apparaît au dénominateur la TFD d'un signal composé des coefficients du filtre complété par des zéros (pour le choix de N, voir paragraphe 1.2.3).

L'ordre du filtre est un paramètre très important : plus l'ordre est grand, plus l'enveloppe reproduira les petites variations du spectre. Un ordre trop faible risque de ne

pas reproduire certaines résonances, un ordre trop élevé reproduira des détails tels que les harmoniques.

Cette technique est bien adaptée aux signaux de voix graves ou pour tout instrument dont la structure formantique est claire et distincte de la structure harmonique. Quand le fondamental est élevé, la LPC sera donc avantageusement remplacée par le cepstre discret.

1.6.3. *Estimation par cepstre discret*

Le cepstre est la TF^{-1} du logarithme du module du spectre du signal analysé. Il est constitué d'un vecteur de coefficients cepstraux notés $\boldsymbol{c} = \{c(k)\}$ pour $k = 0, \ldots, N-1$. Réciproquement, étant donné un vecteur $\boldsymbol{c}$ de coefficients $c(k)_{k=0,\ldots,p}$, l'enveloppe cepstrale correspondante est définie par :

$$ENV(f;\boldsymbol{c}) = c(0) + 2\sum_{k=1}^{p} c(k)\cos(2\pi k f/F_e)$$

pour $f \in [0, F_e/2]$.

Elle correspond au logarithme d'un spectre, l'expression $20ENV(f;\boldsymbol{c})/\log(10)$ est donc comparable à un spectre d'énergie en dB : $20\log_{10}|\tilde{s}|$. Le tracé sur N points s'effectue en remplaçant f par nF_e/N pour $n = 0, \ldots, N-1$.

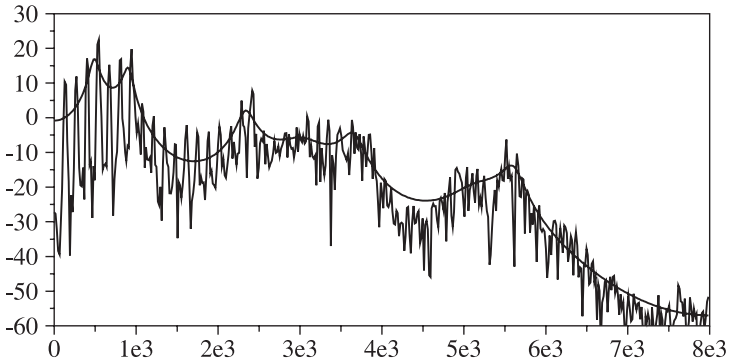


Figure 1.12. *Enveloppe spectrale LPC. Le son d'origine étant une voyelle [o], la figure représente l'enveloppe spectrale superposée à son spectre. Cette enveloppe a été obtenue par l'algorithme de Durbin, le signal ayant été au préalable préaccentué et multiplié par une fenêtre de pondération de Hanning pour le calcul des coefficients d'autocorrélation.*

Pour obtenir l'enveloppe cepstrale d'un signal, il suffit de la calculer sur les premiers coefficients du cepstre : moins il y aura de coefficients, plus l'enveloppe sera

lisse. Cependant, pour des signaux présentant des partiels bien définis, il est préférable d'utiliser le cepstre discret, car il est calculé uniquement sur ces partiels.

L'algorithme est le suivant. Soit les pics spectraux de fréquence $\hat{f}_{l,i}$ et d'amplitude $\hat{A}_{l,i}$, l'enveloppe cepstrale est estimée par minimisation du critère quadratique $err = \sum_{i=1}^{I_l} |\log(\hat{A}_{l,i}) - ENV(\hat{f}_{l,i}; \boldsymbol{c})|^2$ dont la solution directe, obtenue en annulant la différentielle de err par rapport au vecteur $\boldsymbol{c}$, est $\hat{\boldsymbol{c}} = (\boldsymbol{M}^t\boldsymbol{M})^{-1}\boldsymbol{M}^t\boldsymbol{A}$ où $\boldsymbol{A}$ est le vecteur colonne du logarithme des amplitudes $\log(\hat{A}_{l,i})$, $i = 1, \ldots, I_l$ et où $\boldsymbol{M}$ est la matrice à I_l lignes et $p+1$ colonnes de coefficients $\boldsymbol{M}(i,k) = 2\cos(2\pi k \hat{f}_{l,i}/F_e)$, pour $k = 1, \ldots, p$ et $i = 1, \ldots, I_l$ et $\boldsymbol{M}(i,0) = 1$ pour $i = 1, \ldots, I_l$.

La solution obtenue présente en général des oscillations et l'extrapolation (la forme de l'enveloppe en dehors des partiels) est mauvaise. Des techniques de régularisation permettent de supprimer ces inconvénients [CAP 97]. Cette technique présente l'avantage de pouvoir pondérer facilement les différents partiels et comparer différentes enveloppes par distance euclidienne sur les vecteurs de coefficients cepstraux.

1.6.4. *Formants*

Dans le modèle source/filtre, les maximums présentés par l'enveloppe spectrale correspondent à des résonances dues à la partie filtre. La position de ces maximums caractérise en partie le timbre des sons. En particulier pour la voix, ces maximums, appelés formants, caractérisent la voyelle prononcée mais peuvent aussi traduire certains aspects de la qualité de voix comme la perception d'une voix tendue ou relâchée ou certaines techniques de chant comme l'apparition du « formant du chanteur » sur les voix lyriques.

L'estimation de la position des maximums de l'enveloppe spectrale peut être réalisée par une technique de détection de pics analogue à celles développées au paragraphe 1.4.2 pour les partiels. Cependant, lorsque l'enveloppe est déterminée par les coefficients d'un filtre comme dans le cas de la prédiction linéaire, il est possible d'en déduire les pôles (ce sont les zéros du polynôme $\sum_{i=0}^{p} a(i)z^{-i}$) et une approximation de la fréquence des formants grâce au résultat suivant : la fréquence de résonance f_r d'un filtre d'ordre 2 correspondant aux pôles $\rho e^{\pm j\theta}$ vérifie la formule $\cos(2\pi f_r/F_e) = \frac{1}{2}(\rho + 1/\rho)\cos(\theta)$.

1.7. Suivi de notes et applications

Le suivi de notes consiste à transformer le signal acoustique en une représentation musicale du type partition ou plus simplement en signal MIDI. Les informations à extraire sont donc les instants de début et de fin de notes, leurs fréquences et leurs amplitudes.

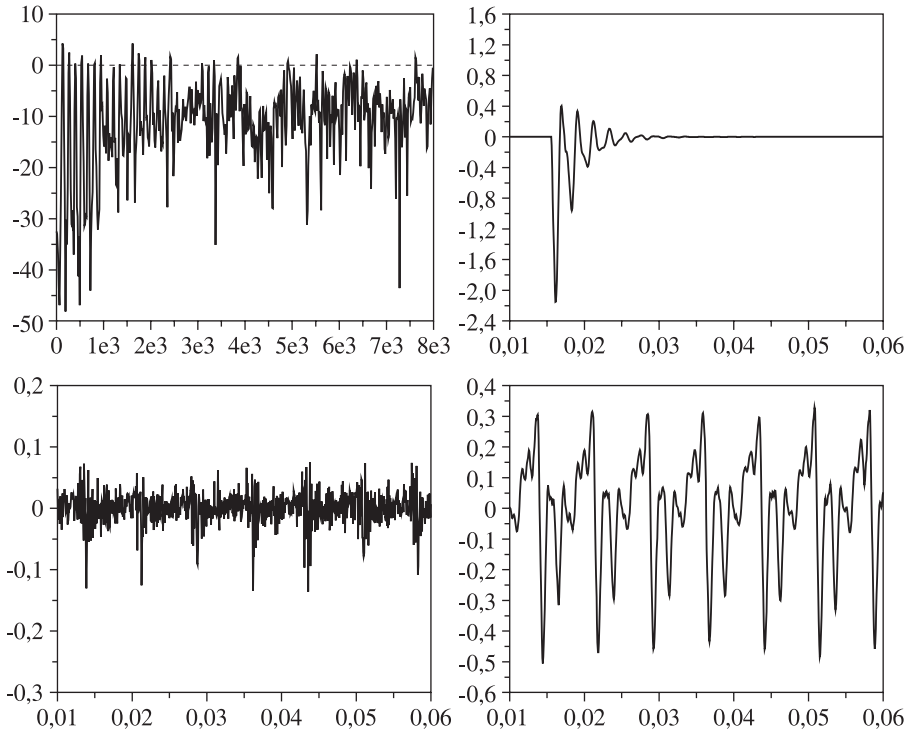


Figure 1.13. *Calcul du résiduel. Sur l'exemple de la figure 1.12, le signal a été filtré par le filtre inverse (correspondant à l'inverse de l'enveloppe spectrale) fournissant le résiduel (en bas à gauche) dont le spectre est plat (en haut à gauche) et qui correspond, en première approximation, à la source sonore. En convoluant ce résiduel par la réponse impulsionnelle du filtre (en haut à droite), on obtient le signal original (en bas à droite). Remarquez que la forme de la réponse impulsionnelle se retrouve sur le signal original à chaque instant où le résiduel contient une impulsion.*

En toute généralité, ce problème est très difficile : comment séparer les différentes notes d'une masse orchestrale ou même d'un petit ensemble ? Comment reconnaître les sons de percussion ? Et même pour un instrument monophonique, la musique contemporaine exploite les possibilités des instruments de telle manière que c'est la notion même de note qui doit être remise en cause. Pour être utilisable, le suivi de note nécessite donc des hypothèses fortes.

Enfin, une difficulté supplémentaire apparaît : la segmentation du signal ne peut pas se faire sur l'amplitude seule à cause des sons filés ou des legatos très lisses, ni sur la fréquence seule à cause des notes répétées. Les algorithmes devront donc combiner les deux sources d'information.

Aux sections 1.3 et 1.5, nous avons présenté des techniques d'estimation de l'amplitude et du fondamental qui fournissent une valeur réelle à chaque instant. Une note MIDI, quant à elle, est décrite par une seule valeur quantifiée d'amplitude (la vélocité) et de fréquence (la hauteur MIDI) pour la durée de la note (les écarts et les variations sont traités à part). Un système de suivi de notes devra donc quantifier la fréquence fondamentale et l'amplitude, et détecter le début et la fin des notes.

1.7.1. *Quantification de la fréquence fondamentale*

La hauteur de note MIDI étant quantifiée par demi-tons, elle s'exprime en fonction de la fréquence en hertz par la formule $q = q_{ref} + Arr(12 \log_2(f/f_{ref}))$, où f_{ref} et q_{ref} sont la fréquence et la hauteur MIDI de référence ($q_{ref} = 69$ pour le diapason standard $f_{ref} = 440$ Hz) et où $Arr(x)$ est l'entier le plus proche de x.

1.7.2. *Détection de début et de fin de notes*

Dans un système développé à l'IRCAM en 1990, l'information des fréquences fondamentales est analysée à court terme ($T_a = 10$ ms) pour déterminer les zones stables, où une zone stable est une suite d'au moins n_0 fréquences fondamentales dont les hauteurs MIDI sont identiques (par défaut $n_0 = 2$). D'autre part, l'information d'amplitude est analysée pour déterminer les attaques et les relâchements. Une attaque est une suite d'amplitudes croissantes supérieures, d'une part, à un seuil de montée relatif à l'amplitude du minimum précédent (d'un facteur $1{,}8$) et, d'autre part, au seuil absolu de bruit (environ -50 dB du maximum). Un relâchement est une suite d'amplitudes décroissantes ayant une vitesse de décroissance supérieure à un seuil de descente (en fait, l'amplitude du point courant doit être inférieure à $0{,}25$ fois le maximum des quatre derniers points) ou ayant une amplitude inférieure au seuil de bruit. L'algorithme prévoit d'envoyer un événement MIDI « note *on* » au début et un « note *off* » à la fin de toute zone d'intersection entre une zone stable et une zone se trouvant entre une attaque et un relâchement. Cet algorithme permet donc de traiter les cas de notes répétées grâce à l'information de changement d'amplitude et aussi les cas de notes legato grâce à l'information de fréquence fondamentale.

Un problème couramment observé est l'apparition de fausses détections dues à la présence de bruit, aux variations internes de fréquences ou d'amplitudes (par exemple le vibrato, difficile à distinguer du trille instrumental) ou même, pour les systèmes multiphoniques, le battement entre des fréquences proches [BOB 98]. Le système décrit ci-dessus prévoit de ne pas répéter de notes si une nouvelle attaque n'est pas générée.

D'autres techniques tentent de segmenter les notes en utilisant plus directement les informations conjointes d'énergie et de fréquence. Bobrek et Koch [BOB 98] proposent un système fondé sur une décomposition du signal en sous-bandes de fréquences. Pour détecter les attaques, ils comptent le nombre de sous-bandes dépassant un seuil adaptatif et décident qu'une note débute à chaque maximum local de ce compteur. La fréquence et l'amplitude de la note ou des notes est obtenue par comparaison entre le contenu fréquenciel des sous-bandes et des valeurs de référence. La fin des notes est effectuée par seuillage absolu. Ce système a été appliqué avec succès sur le suivi de notes de piano.

Goto et Muraoka [GOT 99] détectent les composantes d'attaque par bande de fréquences : en notant $p(l,n) = \frac{1}{N}|\widetilde{s_l}(n)|^2$ la densité spectrale de puissance du signal, si $\min(p(l,n), p(l+1,n)) > \max(p(l-1,n), p(l-1,n \pm 1))$ alors $p(l,n)$ est une composante d'attaque. Les temps d'attaque sont alors calculés comme les pics de la fonction $D(l) = \sum_n d(l,n)$, où les $d(l,n) = \max(p(l,n), p(l+1,n)) - \max(p(l-1,n), p(l-1,n \pm 1))$ sont les vitesses de montée. Cette procédure s'insère dans un système de suivi de pulsation rythmique de musique sans percussion.

Sieger et Tewfik [SIE 98] décomposent le signal en partiels sinusoïdaux selon la méthode de McAulay et Quatieri [MCA 86] (voir section 1.4) et effectuent une reconstruction de la partie sinusoïdale et de la partie bruit. La détection de début et de fin de note est effectuée, d'une part, sur les partiels par regroupement de tous les partiels se superposant, d'autre part, sur la partie bruit par une technique plus classique de seuillage de la dérivée de l'énergie à court terme.

Raphaël [RAP 99] propose de segmenter le signal musical par l'utilisation de modèles de Markov cachés sur l'ensemble du signal, ce qui rend la décision globale et donc permet la prise en considération de la probabilité de transition entre les notes.

1.7.3. *Suivi de partition*

Une des nombreuses applications du suivi de notes est le suivi de partition. Il s'agit de suivre en temps réel sur une partition l'interprétation qu'en donne un musicien. De ce fait, il est possible de programmer le système pour qu'il réponde au musicien : pour jouer un accompagnement tel qu'il est écrit ou en l'adaptant à l'interprétation du musicien, ou encore pour déclencher n'importe quel événement utilisant ou non les sons produits par le musicien.

Dans son principe, le suivi de partition est un problème d'alignement temporel : savoir où le musicien en est dans la partition revient à aligner les notes jouées avec celles qui sont écrites.

La plupart des approches sont fondées principalement sur la mesure de la hauteur de chaque note et seulement en second lieu sur leur durée et leur écart en temps [BAI 93, DAN 84, VER 84]. Certains vont même jusqu'à utiliser l'apparition d'une nouvelle note pour avancer dans la partition, sans tenir compte de l'instant précis où elle devrait être jouée selon cette partition [PUC 92]. Cette approche laisse une grande liberté à l'interprète, mais peut perdre le suivi lors de passages à nombre variable de notes comme dans les trilles ou les notes répétées. A l'opposé, d'autres approches sont fondées essentiellement sur les figures rythmiques et le respect du tempo [VAN 95]. Le suivi simultané des notes jouées et du tempo autorise la prédiction des instants où seront jouées les prochaines notes ; la mesure de l'écart entre la prédiction et la note effective permet de différencier s'il s'agit d'une note de l'accord en cours ou d'une nouvelle note et de recalculer le tempo si nécessaire.

Les difficultés proviennent des erreurs de l'interprète (fausses notes, notes ou groupe de notes pas jouées, notes corrigées après coup), mais aussi des erreurs dues au système de suivi de notes (fausses détections, omissions). La plupart des systèmes prévoient donc des mécanismes de rattrapage pour ces cas d'erreurs. L'utilisation répétée de systèmes de suivi en situation de concert a poussé certains auteurs à développer des mécanismes d'« apprentissage » de certaines caractéristiques de l'interprétation du musicien [VER 85] ou de l'instrument lui-même en contexte musical [KAS 99].

Plus récemment, des systèmes plus performants mais aussi plus complexes de suivi de partitions ont été développés notamment à l'IRCAM [LEM 03, ORI 01a, ORI 01b]. Ils sont construits sur des modèles de Markov cachés à deux niveaux, un niveau représentant la succession des notes de la partition et un niveau représentant l'évolution interne de chaque note. Ces systèmes présentent les avantages de ne plus nécessiter l'estimation directe de paramètres peu robustes (comme le F_0) et d'offrir des possibilités d'apprentissage de l'interprétation musicale.

1.8. Bibliographie

[ALL 77] ALLEN J.B., « Short-term spectral analysis, synthesis and modification by discrete Fourier transform », *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, p. 235-238, 1977.

[BAI 93] BAIRD B., BLEVINS D., ZAHLER N., « Artificial intelligence and music: Implementing an interactive computer performer », *Computer Music Journal*, vol. 17, n° 2, p. 73-79, 1993.

[BLA 98] BLANCHET G., CHARBIT M., *Traitement numérique du signal : simulation sous Matlab*, Hermès, Paris, 1998.

[BOB 98] BOBREK M., KOCH D.B., « Music signal segmentation using tree-structured filter banks », *Journal of the Audio Engineering Society*, vol. 46, n° 5, p. 412-427, 1998.

[BRO 89] BROWN J.C., PUCKETTE M.S., « Calculation of a "narrowed" autocorrelation function », *JASA*, vol. 85, p. 1595-1601, avril 1989.

[CAP 97] CAPPÉ O., OUDOT M., MOULINES E., « Spectral envelope estimation using a penalized likelihood criterion », dans *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, Mohonk, octobre 1997.

[CHA 85] CHAFE C., JAFFE D., KASHIMA K., MONT-REYNAUD B., SMITH III J.O., « Techniques for note identification in polyphonic music », dans *International Computer Music Conference*, 1985.

[CHE 91] DE CHEVEIGNÉ A., « A mixed speech F0 estimation algorithm », *Eurospeech*, vol. 2, p. 445-448, 1991.

[CHE 99] DE CHEVEIGNÉ A., KAWAHARA H., « Multiple period estimation and pitch perception model », *Speech Communication*, vol. 27, n° 3-4, p. 175-185, 1999.

[COO 96] COOPER D., KIA C.N., « A monophonic pitch-tracking algorithm based on waveform periodicity determinations using landmark points », *Computer Music Journal*, vol. 20, n° 3, p. 70-78, 1996.

[DAL 98] D'ALESSANDRO C., DARSINOS V., YEGNANARAYANA B., « Effectiveness of a periodic and aperiodic decomposition method for analysis of voice sources », *IEEE Transactions on Speech and Audio Processing*, vol. 6, n° 1, p. 12-23, janvier 1998.

[DAN 84] DANNENBERG R., « An on-line algorithm for real-time accompaniment », dans *International Computer Music Conference*, San Francisco, Californie, p. 193-198, 1984.

[DEP 93] DEPALLE P., GARCÍA G., RODET X., « Tracking of partials for additive sound synthesis using hidden Markov models », *ICASSP*, vol. 1, p. 225-228, avril 1993.

[DEP 94] DEPALLE P., GARCÍA G., RODET X., « A virtual castrato (!?) », dans *International Computer Music Conference*, Aarhus, Danemark, 1994.

[DOV 94] DOVAL B., Estimation de la fréquence fondamentale des signaux sonores, Thèse de doctorat, Université Paris VI, 1994.

[DUI 82] DUIFHUIS H., WILLEMS L.F., SLYUTER R.J., « Measurement of pitch in speech: An implementation of Goldstein's theory of pitch perception », *JASA*, vol. 71, n° 6, p. 1568-1580, juin 1982.

[FIT 96] FITZ K., HAKEN L., « Sinusoidal modeling and manipulation using Lemur », *Computer Music Journal*, vol. 20, n° 4, p. 44-59, 1996.

[GEO 96] GEOFFROIS E., « The multi-lag-window method for robust extended-range F0 determination », dans *International Conference on Speech and Language Processing*, 1996.

[GOL 69] GOLD B., RABINER L.R., « Parallel processing techniques for estimating pitch periods of speech in the time-domain », *JASA*, vol. 46, p. 442-448, 1969.

[GOT 99] GOTO M., MURAOKA Y., « Real-time beat tracking for drumless audio signals: Chord change detection for musical decisions », *Speech Communication*, vol. 27, n° 3-4, p. 311-335, 1999.

[HAR 78] HARRIS F.J., « On the use of windows for harmonic analysis with the discrete Fourier transform », *Proceedings of the IEEE*, vol. 66, n° 1, p. 51-83, 1978.

[HER 88] HERMES D., « Measurement of pitch by subharmonic summation », *JASA*, vol. 83, n° 1, p. 257-264, janvier 1988.

[HES 83] HESS W., *Pitch Determination of Speech Signals*, Springer-Verlag, Berlin, 1983.

[KAS 99] KASHINO K., MURASE H., « A sound source identification system for ensemble music based on template adaptation and music stream extraction », *Speech Communication*, vol. 27, n° 3-4, p. 337-349, 1999.

[KUH 90] KUHN W.B., « A real-time pitch recognition algorithm for music applications », *Computer Music Journal*, vol. 14, n° 3, p. 60-71, 1990.

[LAN 89] LANSKY P., *Compositional applications of linear predictive coding*, dans Mathews M., Pierce J. (dir.), *Current Directions in Computer Music Research*, MIT Press, Cambridge, Massachusetts, 1989.

[LEM 03] LEMOUTON S., SCHWARZ D., ORIO N., « Score following: State of the art and beyond », dans *Conference on New Instruments for Musical Expression (NIME'03)*, 2003.

[MAH 89] MAHER R.C., An approach for the separation of voices in composite musical signals, Thèse de doctorat, Université de l'Illinois, Urbana-Champain, 1989.

[MAH 90] MAHER R.C., « Evaluation of a method for separating digitized duet signals », *JAES*, vol. 38, n° 12, p. 956-979, décembre 1990.

[MAR 76] MARKEL J.D., GRAY A.H., *Linear Prediction of Speech*, Springer-Verlag, New York, 1976.

[MART 81] MARTIN P., « Extraction de la fréquence fondamentale par intercorrélation avec une fonction peigne », dans *Douzièmes journées d'études sur la parole*, p. 223-232, mai 1981.

[MART 82] MARTIN P., « Comparison of pitch detection by cepstrum and spectral comb analysis », *ICASSP*, vol. 1, p. 180-183, 1982.

[MCA 86] MCAULAY R.J., QUATIERI T.F., « Speech analysis/synthesis based on a sinusoidal representation », *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, p. 744-754, 1986.

[MED 91] MEDAN Y., YAIR E., DAN C., « Super resolution pitch determination of speech signals », *IEEE ASSP*, vol. 39, n° 1, p. 40-48, janvier 1991.

[NOL 67] NOLL A.M., « Cepstrum pitch determination », *JASA*, vol. 41, n° 2, p. 293-309, 1967.

[ORI 01a] ORIO N., DÉCHELLE F., « Score following using spectral analysis and hidden Markov models », dans *ICMC*, ICMA, La Havane, Cuba, 2001.

[ORI 01b] ORIO N., SCHWARZ D., « Alignment of monophonic and polypophonic music to a score », dans *ICMC*, La Havane, Cuba, 2001.

[PAL 83] PALIWAL K.K., RAO P.V.S., « A synthesis-based method for pitch extraction », *Speech Communications*, vol. 2, 1983.

[PAP 84] PAPOULIS A., *Probability, Random Variables and Stochastic Processes*, McGraw-Hill, New York, 1984.

[PIC 98] PICINBONO B., *Signaux et systèmes linéaires*, Ellipses, Paris, 1998.

[POR 80] PORTNOFF M.R., « Discrete time signals and systems based on short-time Fourier analysis », *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, n° 1, p. 55-64, février 1980.

[PUC 92] PUCKETTE M., LIPPE C., « Score following in practice », dans *International Computer Music Conference*, San Francisco, Californie, p. 182-185, 1992.

[RAB 74] RABINER L.R., SCHAFER R.W., « On the behavior of minimax FIR digital Hilbert transformers », *The Bell System Technical Journal*, vol. 53, n° 2, février 1974.

[RAB 75] RABINER L.R., GOLD B., *Theory and application of digital signal processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[RAB 86] RABINER L.R., JUANG B.H., « An introduction to hidden Markov models », *IEEE ASSP Magazine*, p. 4-16, janvier 1986.

[RAP 99] RAPHAEL C., « Automatic segmentation of acoustic musical signals using hidden Markov models », *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, n° 4, p. 360-370, 1999.

[SER 89] SERRA X., A system for sound analysis/transformation/synthesis based on a deterministic plus stochastic decomposition, Thèse de doctorat, CCRMA, Université de Standford, 1989.

[SER 90] SERRA X., SMITH III J.O., « Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition », *Computer Music Journal*, vol. 14, n° 4, p. 12-24, 1990.

[SIE 98] SIEGER N.J., TEWFIK A.H., « Audio coding for representation in MIDI via pitch detection using harmonic dictionaries », *Journal of VLSI Signal Processing Systems for Signal Image and Video Technology*, vol. 20, n° 1-2, p. 45-59, 1998.

[VAN 95] VANTOMME J.D., « Score following by temporal pattern », *Computer Music Journal*, vol. 19, n° 3, p. 50-59, 1995.

[VER 84] VERCOE B., « The synthetic performer in the context of live performance », dans *International Computer Music Conference*, San Francisco, Californie, p. 199-200, 1984.

[VER 85] VERCOE B., PUCKETTE M., « Synthetic rehearsal: Training the synthetic performer », dans *International Computer Music Conference*, San Francisco, Californie, p. 275-278, 1985.

[YEG 98] YEGNANARAYANA B., D'ALESSANDRO C., DARSINOS V., « An iterative algorithm for decomposition of speech signals into periodic and aperiodic components », *IEEE Transactions on Speech and Audio Processing*, vol. 6, n° 1, p. 1-11, janvier 1998.

Chapitre 2

La synthèse additive

Les sons sont des phénomènes physiques appartenant au monde réel, sensible. Afin de synthétiser des sons numériques nouveaux ou de manipuler des sons existants à l'aide d'un ordinateur, il est nécessaire de définir une représentation formelle pour ces sons : un modèle sonore qui doit être aussi général que possible, de sorte que la plupart des sons puissent être fidèlement reproduits et transformés de manière naturelle et expressive musicalement. La modélisation sonore fait le lien entre le monde réel analogique et le monde formel numérique. Le terme synthèse désigne le calcul du signal audio à partir d'un son défini par des paramètres à l'intérieur d'un certain modèle sonore, tandis que l'analyse sonore (voir chapitre « Méthodes d'analyse du signal musical ») consiste en l'extraction des paramètres du modèle à partir de sons réels.

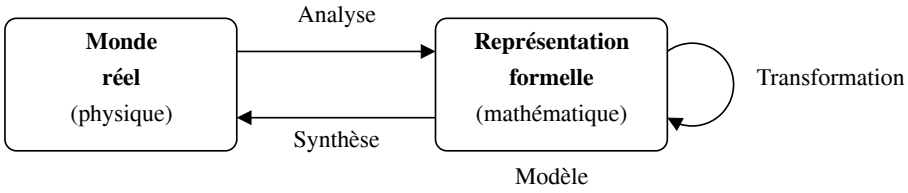
En fait, nous nous contenterons de générer, à l'aide d'algorithmes, la série des échantillons du signal audio s discret, numérique. Le signal continu associé peut alors facilement être reconstruit en temps réel dans le monde réel, physique, en utilisant le convertisseur numérique/analogique (*digital-to-analog converter* ou DAC) d'une carte son par exemple. Le signal discret est caractérisé par le choix d'une fréquence d'échantillonnage F_e. De plus, nous travaillerons avec des nombres à virgule flottante, sans nous soucier de la quantification, c'est-à-dire du nombre de bits nécessaires pour stocker un échantillon.

Il existe une grande quantité de modèles sonores et les techniques de synthèse sont encore plus nombreuses. Dans ce chapitre, nous nous intéresserons principalement au

---

Chapitre rédigé par Sylvain MARCHAND.

modèle additif, décrit en section 2.1, qui est à la base des modèles spectraux. Les principales techniques de synthèse additive en temps réel sont présentées en section 2.2. Les paramètres du modèle additif varient dans le temps et la section 2.3 explique comment contrôler dynamiquement les algorithmes de synthèse présentés. La section 2.4 montre que la prise en considération de phénomènes psycho-acoustiques simples peut aider à gagner du temps lors du processus de synthèse sonore, sans pour autant dégrader la qualité du son perçu. Il existe également des techniques de synthèse dites non linéaires qui permettent de produire à moindre coût des spectres élaborés. Les techniques de synthèse par modulation décrites en section 2.5 réutilisent les notions vues en section 2.2 et sont en fait très proches de la synthèse additive. Un des reproches faits à la synthèse additive et ses dérivés est l'absence de bruit dans les sons générés. La section 2.6 aborde le difficile problème de la synthèse des bruits et montre comment ajouter simplement une composante bruitée à un son trop pur.



La modélisation sonore
entre le monde réel analogique et le monde formel numérique

Afin de bien comprendre ce chapitre, il est nécessaire d'avoir des connaissances de base en mathématiques [MOO 78a, MOO 78b, MOO 85] et traitement du signal [MOOR 85, OPP 89, ORF 96] appliqués à l'informatique musicale.

Les notions élémentaires de traitement du signal et d'algorithmique de niveau deuxième cycle universitaire doivent amplement suffire. Les ouvrages cités précédemment pourront être consultés en complément si nécessaire.

2.1. Le modèle additif

La synthèse additive ([MOOR 77]) est basée sur l'une des toutes premières techniques de modélisation spectrale. Cette dernière repose elle-même sur le théorème de Fourier, qui assure que toute fonction périodique peut être décomposée sous la forme d'une somme de sinusoïdes d'amplitudes diverses et de fréquences liées harmoniquement. Pour les sons pseudo-périodiques, ces amplitudes et ces fréquences varient lentement dans le temps, contrôlant un ensemble d'oscillateurs quasi sinusoïdaux plus communément appelés *partiels* [ASS 85, CAS 94].

2.1.1. *Représentation des sons*

La transformée de Fourier permet de passer de la représentation temporelle (amplitude en fonction du temps) d'un signal sonore à une représentation fréquentielle spectrale (amplitude en fonction de la fréquence). Cette transformée donne en fait l'image fréquentielle du son tout entier et la totalité du signal se retrouve moyennée à l'intérieur d'un seul spectre. Ce spectre coïncide avec notre perception uniquement dans le cas des sons stationnaires. Etant donné que la plupart des sons évoluent dans le temps, les modèles sonores doivent avoir des paramètres qui varient également dans le temps.

Le vocodeur de phase [DOL 86, MOOR 78, SER 97] utilise la transformée de Fourier à court terme [ALL 77a, ALL 77b, POR 76, POR 80]. Cette version de la transformée de Fourier est dépendante du temps. En fait, une fenêtre d'analyse se déplace dans le son [HAR 78] et, à intervalles réguliers, le signal est pondéré (multiplié) par la fenêtre ; la transformée de Fourier classique est effectuée sur la portion de signal correspondante. On parcourt ainsi la totalité du son tout en produisant une série de spectres à court terme pris pour de petites portions du signal à des instants différents (voir chapitre « Méthodes d'analyse du signal musical »).

L'analyse de McAulay-Quatieri [MCA 86] suit les évolutions des maximums locaux (pics) à l'intérieur des spectres à court terme pour reconstruire les évolutions des partiels dans le temps. Ces partiels correspondent à des oscillations quasi sinusoïdales caractérisées par des fréquences et des amplitudes instantanées qui varient lentement et continûment dans le temps. Le signal audio a peut alors être calculé à partir des paramètres additifs en utilisant les équations suivantes :

$$a(t) = \sum_{p=1}^{P} a_p(t) \cos\big(\phi_p(t)\big) \qquad [2.1]$$

$$\frac{d\phi_p}{dt}(t) = 2\pi f_p(t) \;\text{ c'est-à-dire }\; \phi_p(t) = \phi_p(0) + 2\pi \int_0^t f_p(u)\, du \qquad [2.2]$$

où P est le nombre de partiels et les fonctions f_p, a_p et ϕ_p sont respectivement la fréquence, l'amplitude et la phase instantanées du p-ième partiel. Les P paires (f_p, a_p) sont les paramètres du son dans le modèle additif et définissent à l'instant t des points dans le plan fréquence-amplitude, comme indiqué sur la figure 2.1. Les évolutions de ces partiels dans le temps forment des courbes comme dans la figure 2.2. Cette représentation est utilisée dans de nombreux programmes d'analyse/synthèse comme Lemur [FIT 96], SMS [SERR 97] ou *InSpect* [MAR 99b, MAR 00a, MAR 01a]. Il est important de noter que les phases des partiels sont recalculées et non pas issues de l'analyse. Plus précisément, ϕ_p peut être recalculée en utilisant la phase initiale $\phi_p(0)$ et l'équation [2.2]. La phase initiale $\phi_p(0)$ peut aussi être ignorée durant la phase d'analyse et prendre une valeur arbitraire (par exemple $\pi/2$) lors de la resynthèse.

Ceci est la conséquence d'expérimentations psycho-acoustiques [CAB 76, HEL 54, RIS 91] qui sortent du cadre de ce chapitre. Mais ceci explique que dans la suite nous ne prêterons guère attention à la phase.

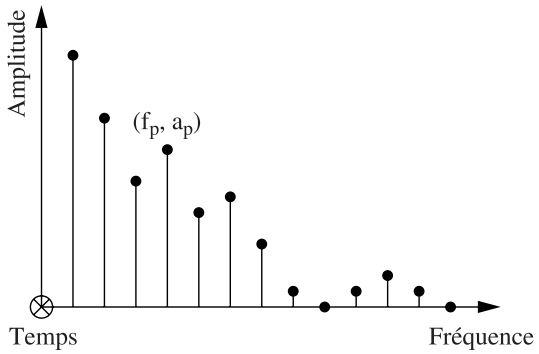


Figure 2.1. *Les partiels d'un son harmonique à l'instant t*

Notons également que la fonction sinus peut être utilisée à la place de la fonction cosinus dans l'équation [2.1], puisque ces deux fonctions sont en fait les mêmes à un déphasage de $\pi/2$ près :

$$\sin(x) = \cos\left(x - \tfrac{\pi}{2}\right) \qquad [2.3]$$

$$\cos(x) = \sin\left(x + \tfrac{\pi}{2}\right) \qquad [2.4]$$

2.1.1.1. *Evolution des partiels dans le temps*

L'hypothèse que les fréquences et les amplitudes des partiels varient *lentement* dans le temps est vraiment très importante. Par exemple, pour un son a donné, la solution $P = 1$, $f_1(t) = 0$, $a_1(t) = a(t)$ vérifie trivialement les équations du modèle, mais comme a_1 ne varie pas lentement dans le temps cette solution n'est pas acceptable. Cependant, la définition de « varier lentement dans le temps » se doit d'être clarifiée. Formellement, pour tout partiel p, les variations des fonctions a_p et f_p sont limitées en fréquence par une fréquence faible, située bien en dessous de la plus petite des f_p. Plus précisément, ces fonctions ont des spectres limités en fréquence par une fréquence $F_{\max}$ autour de 20 Hz (la plus petite fréquence audible). En effet, les variations des paramètres du modèle doivent rester inaudibles, sous peine de voir les phénomènes de modulation apparaître (voir section 2.5) et remettre en cause la cohérence perceptive des paramètres. Par conséquent, le célèbre théorème d'échantillonnage de Shannon-Nyquist [NYQ 28, SHA 49] assure qu'en théorie échantillonner les évolutions des partiels à deux fois $F_{\max}$ échantillons par secondes est suffisant. En fait, des tests psycho-acoustiques montrent que la fréquence maximale du signal

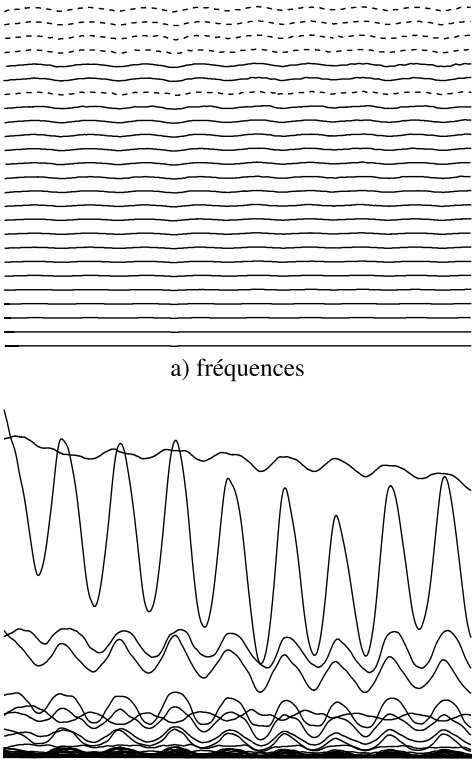


Figure 2.2. *Les évolutions des partiels d'un saxophone alto pendant une seconde. Les fréquences (a) et les amplitudes (b) sont représentées (en ordonnées) en fonction du temps (en abscisse)*

de contrôle $F_{\max}$ ne devrait pas être une constante, mais une fonction proportionnelle à la fréquence du partiel contrôlé. Plus précisément, pour éviter les phénomènes de modulation, $F_{\max}$ est d'environ 0,35 % de la fréquence du partiel contrôlé, c'est-à-dire 70 Hz pour 22 kHz (la plus grande fréquence audible).

2.1.1.2. *Amplitude*

L'amplitude (globale) est aussi une fonction du temps t. Les êtres humains perçoivent l'amplitude sur une échelle logarithmique, l'amplitude étant liée à l'intensité qui correspond au volume en dB. Dans la représentation additive vue précédemment, l'amplitude A correspond à la somme des amplitudes des partiels et peut être calculée à partir des paramètres additifs en utilisant l'équation [2.5]. Afin de considérer l'amplitude RMS (*Root Mean Square*), plus proche de la perception, l'équation [2.6] doit

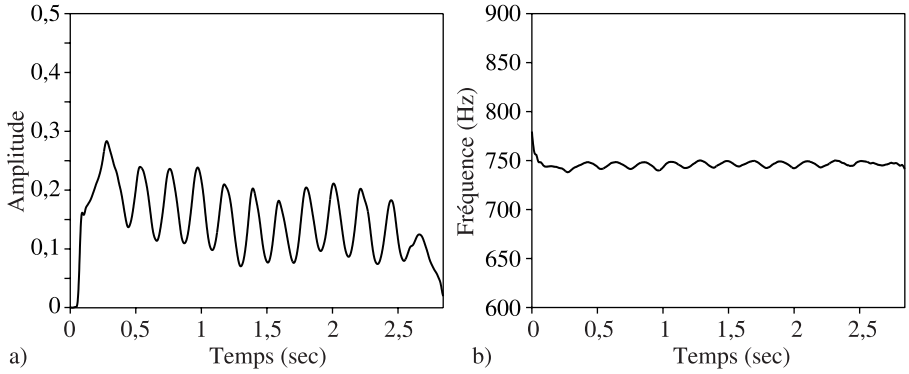


Figure 2.3. *Exemple d'évolutions d'un partiel en amplitude (a) et en fréquence (b)*

être utilisée au lieu de l'équation [2.5] :

$$A(t) = \sum_{p=1}^{P} a_p(t) \qquad [2.5]$$

$$A_{\text{RMS}}(t) = \frac{1}{\sqrt{2}} \sqrt{\sum_{p=1}^{P} \big(a_p(t)\big)^2} \qquad [2.6]$$

Quand le son est amplifié, ces deux amplitudes ont les mêmes variations relatives :

$$\begin{aligned} \forall p, \quad a_p &\to k \cdot a_p \\ A &\to k \cdot A \\ A_{\text{RMS}} &\to k \cdot A_{\text{RMS}} \end{aligned}$$

Et puisque seules les valeurs relatives comptent pour l'oreille, n'importe laquelle de ces deux amplitudes peut être utilisée. Toutefois, l'amplitude RMS est plus proche de l'intensité réellement perçue.

Il est facile de calculer le volume en dB à partir de l'amplitude :

$$V(A) = 20 \log_{10}\Big(\frac{A}{A_{0\,\text{dB}}}\Big)\ \text{dB} \qquad [2.7]$$

où $A_{0\,\text{dB}}$ est l'amplitude de référence correspondant à 0 dB. Dans la suite de cette section, nous utiliserons les valeurs $1/\sqrt{2}$ ou 1 selon que nous considérerons ou non l'amplitude RMS.

2.1.1.3. *L'amplitude RMS*

En fait, l'équation [2.6] est seulement une approximation de la véritable amplitude RMS a_{RMS}, mesurée pour un signal s donné à partir de sa représentation temporelle :

$$a_{\text{RMS}}[s] = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(s[k])^2} \qquad [2.8]$$

Cependant, il s'agit vraiment d'une très bonne approximation. La raison en est la suivante. Pour un partiel p de fréquence f, d'amplitude a et de phase initiale ϕ_0, nous avons :

$$a_{\text{RMS}}[a\sin(2\pi f T_s + \phi_0)] = \frac{a}{\sqrt{2}} \qquad [2.9]$$

Notons que l'amplitude RMS ne dépend pas de la phase. C'est encore vrai quand le son est composé d'une somme de partiels, à condition toutefois que leurs fréquences soient différentes. Pour deux sons quelconques, nous avons :

$$a_{\text{RMS}}[s_1 + s_2] = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(s_1[k] + s_2[k])^2}$$

$$a_{\text{RMS}}[s_1 + s_2] = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\big((s_1[k])^2 + (s_2[k])^2 + 2s_1[k]s_2[k]\big)}$$

$$a_{\text{RMS}}[s_1 + s_2] = \sqrt{\frac{1}{N}\left(\sum_{k=1}^{N}(s_1[k])^2 + \sum_{k=1}^{N}(s_2[k])^2 + 2\sum_{k=1}^{N}s_1[k]s_2[k]\right)}$$

Considérons alors des sons composés d'un unique partiel dans le modèle additif. Pour deux sons simples de ce type s_1 et s_2 de fréquences différentes, les termes croisés de l'équation ci-dessus peuvent être négligés. Plus formellement, $\sum_{k=1}^{N} s_1[k]s_2[k] \approx 0$ car $\sum_{k=1}^{N}\sin(c_1 k)\sin(c_2 k) \approx 0$ pour $c_1 \neq c_2$. Il s'agit en fait du même argument mathématique que celui utilisé dans la décomposition de Fourier. Par conséquent, nous avons :

$$(a_{\text{RMS}}[s_1 + s_2])^2 \approx (a_{\text{RMS}}[s_1])^2 + (a_{\text{RMS}}[s_2])^2$$

et finalement, puisque l'amplitude est toujours positive :

$$a_{\text{RMS}}[s_1 + s_2] \approx \sqrt{(a_{\text{RMS}}[s_1])^2 + (a_{\text{RMS}}[s_2])^2} \qquad [2.10]$$

La généralisation de l'équation [2.10] pour n sons comportant exactement un seul partiel conduit, par induction, à l'équation [2.6].

2.1.1.4. *Fréquence fondamentale*

Pour une simple sinusoïde, la hauteur perçue est directement liée à la fréquence de la sinusoïde. Pour les sons harmoniques, les fréquences des partiels (alors appelés harmoniques) sont multiples d'une fréquence F appelée fréquence fondamentale, qui est une fonction du temps t. C'est de cette fréquence que se dégage la sensation de hauteur, perçue également sur une échelle logarithmique. Par exemple, la hauteur (*pitch* en anglais) MIDI [MIDI 96] est donnée par l'équation suivante :

$$H(F) = 69 + 12 \log_2\Big(\frac{F}{440\,\text{Hz}}\Big) \qquad [2.11]$$

où 69 correspond au do de l'octave 3, 70 au do dièse, etc.

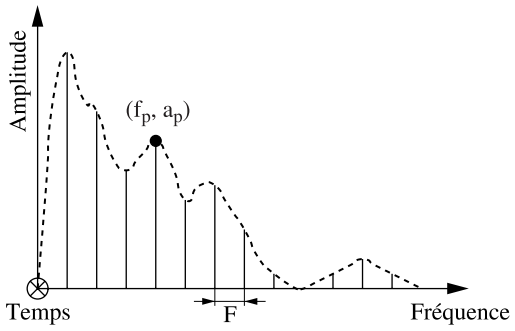


Figure 2.4. *Un son harmonique à l'instant t, de fréquence fondamentale F. L'enveloppe spectrale est indiquée en pointillés*

2.1.1.5. *Enveloppe spectrale*

L'enveloppe spectrale [RIS 86, SCHW 99] joue un rôle très important dans la notion de timbre. Il s'agit, à l'instant t, d'une courbe continue passant par les points discrets $(f_p(t), a_p(t))$ définis par les P partiels, comme indiqué sur la figure 2.4. Cette courbe varie dans le temps et traduit à chaque instant les rapports d'amplitude entre les différents partiels du son.

2.1.2. *Transformations sonores*

Un modèle sonore n'est utile musicalement que s'il peut représenter un nombre suffisant de sons tout en offrant la possibilité de transformer musicalement ces sons. Le modèle additif fournit une représentation du son favorisant des transformations sonores intuitives et musicalement expressives [MAR 99a, MAR 01b].

2.1.2.1. *Etirement temporel* (time-stretching)

Alors que modifier la durée d'un son sans changer sa hauteur présente une réelle difficulté dans le modèle temporel classique [FED 98], cette opération est très facile à réaliser dans le modèle additif. C'est tout simplement une affaire d'échelle sur l'axe du temps. Plus précisément, il suffit de considérer les fonctions $(f_p(kt), a_p(kt))$ au lieu de $(f_p(t), a_p(t))$ pour tout partiel p du son, ce qui conduit en pratique à un rééchantillonnage des évolutions des partiels dans le temps. Le son obtenu est k fois plus court que le son d'origine. La durée du son augmente (respectivement diminue) pour $k < 1$ (respectivement $k > 1$). Bien entendu, k n'est pas nécessairement une constante et peut également varier dans le temps, ce qui peut produire des effets musicaux tout à fait étonnants.

2.1.2.2. *Transposition* (pitch-shifting)

Modifier la hauteur d'un son sans changer sa durée est une opération difficile à réaliser dans le modèle temporel classique [BRI 95], mais pas dans le modèle additif. Une première possibilité est tout simplement de multiplier par k toutes les fréquences f_p des partiels. Le facteur de transposition en octaves est le logarithme en base 2 de k. Si k est une constante dans le temps, l'effet obtenu est une simple transposition. Si k est une sinusoïde avec une fréquence autour de 8 Hz, l'effet musical obtenu est un *vibrato* de cette fréquence. Si les variations de k sont lentes et mathématiquement monotones, on obtient un *glissando* ou un *portamento*. Le problème avec cette première possibilité est que l'enveloppe spectrale se trouve modifiée, ce qui est très gênant dans le cas de la voix chantée par exemple. Si en plus on tient compte de l'enveloppe spectrale et que l'on modifie également les amplitudes des partiels a_p pour conserver cette enveloppe, alors l'effet de transposition obtenu est beaucoup plus réaliste. La figure 2.5 illustre la transposition du son de la figure 2.4 à l'octave inférieure ($k = 0{,}5$) en conservant l'enveloppe spectrale.

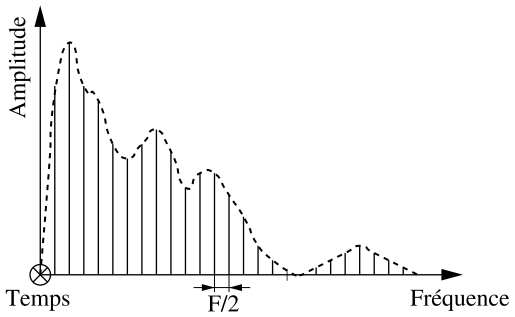


Figure 2.5. *Le son de la figure 2.4 transposé à l'octave inférieure (mêmes enveloppes spectrales, mais fréquences fondamentales différentes)*

2.1.2.3. *Amplification*

Changer le volume d'un son est trivial : il suffit de multiplier par k les amplitudes de tous les partiels. Si k est une constante, l'effet obtenu est une simple amplification. Si k est une sinusoïde avec une fréquence autour de 8 Hz, l'effet musical obtenu est un *tremolo* de cette fréquence. Si les variations de k sont lentes et mathématiquement monotones, il s'agit de fondus : *fade-in* si k est une fonction croissante du temps ou *fade-out* si k est une fonction décroissante.

2.1.2.4. *Filtrage*

La réponse fréquentielle d'un filtre est en fait une enveloppe spectrale F. En multipliant, pour chaque partiel p, l'amplitude a_p par le gain du filtre à la fréquence correspondante, $F(f_p)$, on réalise un filtrage parfait (sans artefacts) du son d'origine par le filtre F. Quand en plus F varie dans le temps, on obtient des effets spéciaux très utilisés actuellement dans les musiques dites électroniques.

2.1.2.5. *Distorsion fréquentielle* (frequency warping)

Quand les sons ne sont pas parfaitement harmoniques [MAT 80a, MAT 80b], comme pour les gongs ou les cloches par exemple, les fréquences des partiels ne sont pas exactement multiples de la fréquence fondamentale F. Il peut être intéressant, pour un son harmonique, de déplacer légèrement les fréquences des partiels par rapport à leur position harmonique idéale. Ceci constitue un effet spécial de distorsion fréquentielle (*warping* en anglais, voir [EVA 98]).

2.1.2.6. *Synthèse croisée*

Il est possible de produire des sons hybrides en conservant la structure harmonique d'un premier son tout en lui appliquant l'enveloppe spectrale d'un second. L'effet obtenu est appelé synthèse croisée. Il s'agit en fait de conserver les fréquences des partiels du premier son, tout en modifiant leurs amplitudes en accord avec l'enveloppe spectrale du second son.

2.1.2.7. *Morphing*

Au lieu de remplacer totalement une caractéristique d'un son par celle d'un autre son, il est aussi possible de mélanger ces caractéristiques. L'effet obtenu est appelé *morphing*. Il en existe autant de sortes que de façons de mélanger les caractéristiques des sons. Par exemple, si l'on considère deux sons harmoniques définis par leurs amplitudes, leurs fréquences fondamentales et leurs enveloppes spectrales, on peut par exemple réaliser entre deux sons une interpolation linéaire de ces paramètres exprimés dans une échelle logarithmique. Ce *morphing* donne des résultats satisfaisants pour les sons instrumentaux, mais pas pour la voix chantée car les formants (maximums locaux dans les enveloppes spectrales) ne sont pas respectés.

En fait, le modèle additif ouvre un vaste champ d'investigation pour les effets sonores.

2.1.3. *Modèles dérivés*

Le modèle additif vu précédemment peut fidèlement reproduire une grande variété de sons pseudo-périodiques, sans bruit et sans transitoires toutefois.

2.1.3.1. Spectral Modeling Synthesis (SMS)

Le modèle *Spectral Modeling Synthesis* (SMS) ajoute une composante bruitée au modèle additif de base, purement sinusoïdal. Serra et Smith proposent dans [SERR 89, SERR 90] un modèle spectral basé sur une décomposition du signal sonore en termes de parties déterministe (sinusoïdes) et stochastique (bruit). La partie déterministe consiste en une somme de partiels, exactement comme le modèle additif classique vu précédemment. Lors de l'analyse, SMS cherche les paramètres amplitude, fréquence et phase de chaque partiel (voir chapitre « Méthodes d'analyse du signal musical »). Puis, il synthétise la partie déterministe en tenant compte des phases des partiels (voir section 2.2) et soustrait alors du signal original le résultat obtenu. La partie stochastique est ce qui reste après cette soustraction. En théorie, il s'agit d'un signal aléatoire que SMS choisit de modéliser sous la forme de bruit blanc filtré par une enveloppe spectrale et qui peut être synthétisé en utilisant la technique présentée dans la section 2.6.

2.1.3.2. Sinusoids + Transients + Noise (S + T + N)

Le modèle SMS peut reproduire des sons pseudo-périodiques avec du bruit mais sans transitoires. Verma et Meng présentent dans [VER 98a, VER 98b] le modèle *Sinusoids + Transients + Noise* (S + T + N) qui étend le modèle spectral SMS avec des transitoires (*transients* en anglais) représentées sous la forme temporelle classique. La difficulté avec ce modèle hybride est de définir une méthode d'analyse capable de séparer avec précision les parties sinusoïdale, bruitée et les transitoires. Le principe général est de localiser les petites zones de variation rapide d'énergie (les transitoires) à l'intérieur de la partie stochastique de SMS, puis de les séparer du bruit proprement dit.

2.2. Synthèse des partiels en temps réel

L'intérêt pratique du modèle additif décrit dans la section 2.1 serait très réduit sans une méthode de synthèse efficace, capable de générer le signal audio à partir des paramètres du modèle, et ce si possible en temps réel.

Dans cette section, nous nous intéressons à la synthèse additive en temps réel. Nous disposons en entrée d'un ensemble de partiels dont les fréquences et les amplitudes varient lentement dans le temps. Ces paramètres contrôlent les fréquences et les amplitudes d'un ensemble d'oscillateurs. La génération du son consiste à calculer les valeurs instantanées de tous les oscillateurs et d'additionner les résultats obtenus.

2.2.1. *Oscillateurs logiciels*

Du fait de l'équation [2.1], la synthèse additive nécessite le calcul d'un grand nombre d'oscillations sinusoïdales. Chaque oscillation est produite par un oscillateur dont la fréquence f et l'amplitude a varient lentement dans le temps (voir figure 2.6). Concentrons-nous alors sur la synthèse d'un seul de ces oscillateurs. Plus formellement, il nous faut générer la séquence d'échantillons suivante :

$$s[n] = a\cos(n\Delta_\phi + \phi_0) \quad \text{où} \quad \Delta_\phi = \frac{2\pi f}{F_e} \qquad [2.12]$$

Le problème principal est alors de calculer la valeur $s[n]$ de chaque oscillateur le plus efficacement possible.

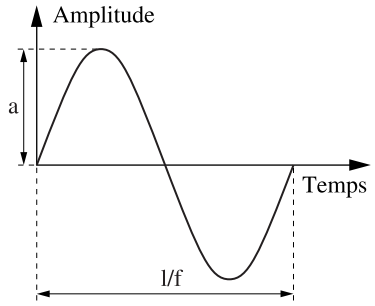


Figure 2.6. *Une période d'une oscillation de fréquence f et d'amplitude a*

Bien entendu, il n'est pas possible de calculer directement la fonction cosinus elle-même pour chaque échantillon. En effet, cela donnerait une synthèse très précise mais extrêmement lente. En fait, même la fonction $\cos$ fournie par les coprocesseurs arithmétiques des processeurs actuels est trop lente pour nos objectifs. Cette fonction peut éventuellement être appelée de temps en temps, mais certainement pas à chaque échantillon.

Le problème est alors de trouver une méthode très rapide pour générer la séquence d'échantillons pour chaque oscillateur, avec un nombre d'opérations aussi réduit que possible. Essentiellement deux possibilités s'offrent alors à nous. La première est basée sur une table d'onde et la seconde sur le calcul incrémental de chaque échantillon à partir de quelques échantillons précédents.

Le principe de la table d'onde est de précalculer un certain nombre de valeurs de la fonction sinus, de les stocker dans un tableau et d'y accéder en utilisant un indice (représentant le temps) lors de l'étape de synthèse proprement dite. Cependant, le principe de la table d'onde requiert au mininum une addition et une multiplication

par échantillon généré, respectivement pour mettre à jour l'indice pour l'échantillon suivant et pour amplifier l'oscillation précalculée pour l'amener à l'amplitude voulue. Afin d'obtenir une qualité acceptable, il faut en plus soit interpoler entre les différentes valeurs stockées dans la table (ce qui prend du temps machine), soit utiliser une table très grande (ce qui consomme de l'espace mémoire).

Par le passé, c'est la méthode de la table d'onde qui était systématiquement retenue, car elle était sensiblement plus rapide. En effet, les accès à la mémoire vive étaient relativement rapides en comparaison avec les opérations arithmétiques, beaucoup plus lentes, et tout particulièrement pour les nombres à virgule flottante. Les processeurs gagnant rapidement en efficacité, alors que la rapidité de la mémoire vive restait pratiquement à la même, cette technique devint moins intéressante. En même temps, des progrès conséquents furent réalisés pour l'accélération des opérations arithmétiques en virgule flottante. De nos jours, un processeur peut effectuer une addition en virgule flottante en seulement un cycle (en utilisant le mécanisme du *pipeline*) avec un temps de latence de trois cycles environ et une multiplication en virgule flottante peut être réalisée en un ou deux cycles avec une latence autour de cinq cycles [1]. La tendance pour les processeurs est actuellement à des opérations arithmétiques et des fréquences internes toujours plus rapides, alors que la vitesse de la mémoire semble stagner.

Un choix naturel semble alors de s'intéresser aux méthodes basées sur un calcul incrémental avec arithmétique en virgule flottante en utilisant un algorithme récursif.

2.2.1.1. *La « forme couplée »*

Considérons des nombres complexes dans le plan complexe et notons $(x[n], y[n]) = x[n] + i\,y[n]$. Par conséquent, la notation d'Euler est $(\cos(\phi), \sin(\phi)) = e^{i\,\phi}$, et nous avons $s[n] = \mathrm{Re}(V_n)$ avec $V_n = a\,e^{i(n\Delta_\phi + \phi_0)}$, où $\mathrm{Re}(x)$ est la partie réelle du nombre complexe x. La figure 2.7 montre le cercle trigonométrique dans le plan complexe. La rotation par Δ_ϕ du vecteur V_n est équivalente à la multiplication complexe du nombre complexe V_n par $e^{i\,\Delta_\phi}$. Plus formellement :

$$V_{n+1} = a\,e^{i((n+1)\Delta_\phi + \phi_0)} = a\,e^{i(n\Delta_\phi + \phi_0)}\,e^{i\,\Delta_\phi} = V_n \cdot e^{i\,\Delta_\phi}$$

Cette équation est la clef de l'algorithme de la « forme couplée » (*coupled form* en anglais, voir [GOR 85, SMI 92]), qui peut être défini par le système d'équations suivant :

$$\begin{cases} (x[0], y[0]) = \big(\cos(\phi_0), \sin(\phi_0)\big) \\ (C_x, C_y) = \big(\cos(\Delta_\phi), \sin(\Delta_\phi)\big) \\ (x[n+1], y[n+1]) = \big(x[n], y[n]\big) \cdot \big(C_x, C_y\big) \end{cases} \qquad [2.13]$$

1. Ces chiffres sont donnés pour la famille des Intel Pentium.

c'est-à-dire :

$$\begin{cases} x[n+1] = x[n] \cdot C_x - y[n] \cdot C_y \\ y[n+1] = x[n] \cdot C_y + y[n] \cdot C_x \end{cases}$$

Le calcul itératif de chaque échantillon $s[n] = x[n]$ de l'oscillateur nécessite quatre multiplications et deux additions. Notons que ce calcul peut être numériquement instable s'il est effectué en arithmétique à virgule fixe.

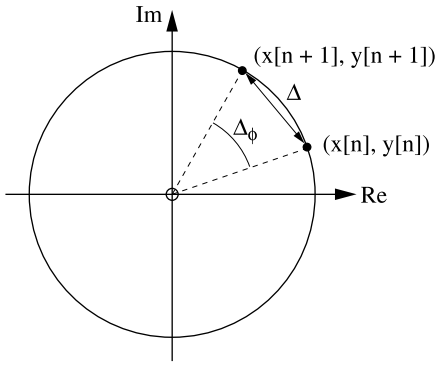


Figure 2.7. *Le cercle trigonométrique : V_{n+1} résulte de la rotation du vecteur V_n par Δ_ϕ*

2.2.1.2. *Le « cercle magique »*

L'algorithme du « cercle magique » (*magic circle* en anglais, voir [GOR 85, SMI 92]) est numériquement stable et nécessite seulement deux multiplications et deux additions par échantillon. Pour $\phi_0 = \pi/2$, il est facile de vérifier par induction, puisque $\sin(2u) = 2\sin(u)\cos(u)$, que les échantillons de s sont donnés par x dans le système d'équations suivant (pour $\phi_0 = 0$ et en utilisant la fonction sinus au lieu du cosinus dans l'équation [2.12] cependant) :

$$\begin{cases} x[0] = 0 \\ y[0] = \cos(\Delta_\phi/2) \\ C = 2\sin(\Delta_\phi/2) \\ x[n+1] = x[n] + C \cdot y[n] \\ y[n+1] = y[n] - C \cdot x[n+1] \end{cases} \qquad [2.14]$$

De nombreuses formules ont été proposées par le passé pour réduire le nombre de multiplications et d'additions nécessaires. Elles tirent toutes avantage de formules trigonométriques bien connues afin de réduire la complexité des calculs. Cependant,

les méthodes qui en découlent utilisent seulement la valeur actuelle de l'oscillateur pour calculer la suivante. Une excellente idée est alors de tirer parti d'un schéma de récursion plus profond.

2.2.1.3. *Le « résonateur numérique »*

Smith utilise le « résonateur numérique » (*digital resonator* en anglais, voir [GOR 85, SMI 92]) afin de réaliser le calcul incrémental de la fonction sinus en utilisant les deux valeurs calculées précédemment, pour en déduire la nouvelle valeur à l'aide de seulement une multiplication et une addition. Cet algorithme est optimal en termes de complexité. Il est en effet impossible d'obtenir de meilleurs résultats, puisqu'une addition sans multiplication (respectivement une multiplication sans addition) produirait seulement une progression arithmétique (respectivement géométrique), bien éloignée de la sinusoïde attendue :

$$s[n] = a\cos(n\Delta_\phi + \phi_0) \quad \text{où} \quad \Delta_\phi = \frac{2\pi f}{F_e}$$

Par conséquent, nous avons :

$$\begin{aligned} s[n+1] &= a\cos\big((n+1)\Delta_\phi + \phi_0\big) \\ &... = a\cos(n\Delta_\phi + \phi_0 + \Delta_\phi) \\ &... = a\cos(n\Delta_\phi + \phi_0)\cos(\Delta_\phi) - a\sin(n\Delta_\phi + \phi_0)\sin(\Delta_\phi) \end{aligned}$$

et :

$$\begin{aligned} s[n-1] &= a\cos\big((n-1)\Delta_\phi + \phi_0\big) \\ &... = a\cos(n\Delta_\phi + \phi_0 - \Delta_\phi) \\ &... = a\cos(n\Delta_\phi + \phi_0)\cos(\Delta_\phi) + a\sin(n\Delta_\phi + \phi_0)\sin(\Delta_\phi) \end{aligned}$$

donc :

$$\begin{aligned} s[n+1] + s[n-1] &= 2a\cos(n\Delta_\phi + \phi_0)\cos(\Delta_\phi) \\ &... = 2\cos(\Delta_\phi)s[n] \\ &... = C\,s[n] \end{aligned}$$

où $C = 2\cos(\Delta_\phi)$.

Finalement, l'algorithme du « résonateur numérique » peut être résumé par le système d'équations suivant :

$$\begin{cases} s[0] = a\cos(\phi_0) \\ s[1] = a\cos(\Delta_\phi + \phi_0) \\ C = 2\cos(\Delta_\phi) \\ s[n+1] = C \cdot s[n] - s[n-1] \end{cases} \qquad [2.15]$$

Pour un oscillateur, le calcul incrémental de chaque échantillon nécessite seulement une multiplication et une addition, mais peut se révéler numériquement instable s'il est effectué en arithmétique à virgule fixe. Nous recommandons par conséquent d'utiliser l'arithmétique à virgule flottante.

2.2.2. *Utilisation de la transformée de Fourier inverse*

Afin de synthétiser un grand nombre de sinusoïdes simultanément, Freed, Rodet et Depalle proposent dans [FRE 92, FRE 93] d'utiliser la transformée de Fourier inverse, sous la condition que les paramètres des partiels varient extrêmement lentement. L'idée est de reconstruire le spectre à court terme du son à l'instant t, pour ensuite lui appliquer la transformée de Fourier rapide inverse (en anglais *inverse fast Fourier transform*, notée IFFT ou FFT^{-1}) afin d'obtenir la représentation temporelle du son, puis de répéter la même opération à l'instant suivant, et ainsi de suite, en se comportant comme une sorte de « vocodeur de phase inverse ».

2.2.2.1. *Rappels sur les transformées de Fourier*

La transformée de Fourier et son inverse sont des transformations mathématiques qui permettent de passer, respectivement, du domaine temporel au domaine fréquentiel et inversement.

Si s et S sont les expressions d'un même signal respectivement dans les domaines temporel et fréquentiel, alors la transformée de Fourier continue et son inverse sont définies par les équations suivantes :

$$S(f) = \int_{-\infty}^{+\infty} s(t)\, e^{-i\,2\pi f t}\, dt \qquad [2.16]$$

$$s(t) = \int_{-\infty}^{+\infty} S(f)\, e^{+i\,2\pi f t}\, df \qquad [2.17]$$

où f est la fréquence exprimée en hertz (Hz) et t le temps en secondes (s).

En pratique, les signaux considérés ne sont pas continus mais discrets, c'est-à-dire échantillonnés dans le temps avec une période d'échantillonnage T_e. Nous noterons alors :

$$s[k] = s(k\,T_e) \qquad [2.18]$$

La fréquence d'échantillonnage F_e (exprimée en Hz) est l'inverse de la période T_e (exprimée en s).

Si le signal discret s est de durée finie N, nous utilisons la transformée de Fourier discrète (TFD) et son inverse. Définissons au préalable :

$$f_m = \frac{mF_e}{N} \qquad [2.19]$$

Les valeurs f_m pour $m \in [0, N-1]$ sont les fréquences de la TFD. Si nous notons $S[m] = S(f_m)$, alors la TFD et son inverse sont définies par les équations suivantes :

$$S[m] = \sum_{n=0}^{N-1} s[n] e^{-i\,2\pi \frac{mn}{N}} \qquad [2.20]$$

$$s[k] = \frac{1}{N} \sum_{n=0}^{N-1} S[n] e^{+i\,2\pi \frac{kn}{N}} \qquad [2.21]$$

Il est parfois commode d'utiliser un facteur de normalisation $2/N$ dans l'équation [2.20] afin de lire l'amplitude des sinusoïdes directement dans le spectre. Dans ce cas, $1/2$ est utilisé au lieu de $1/N$ dans l'équation [2.21].

Les spectres résultants sont constitués de nombres complexes. Les spectres de puissance et de phase correspondent respectivement aux modules et aux arguments de ces valeurs complexes. Un signal s est réel si et seulement si son spectre S est conjugué-symmétrique (partie réel paire, partie imaginaire impaire), comme indiqué dans la figure 2.8.

2.2.2.2. *Approximation du spectre à court terme*

La première étape de l'algorithme consiste à reconstruire le spectre à court terme du son à l'instant t, et ce à partir de l'information spectrale contenue dans les partiels du son. En fait, l'effet de chaque partiel doit être reproduit dans le spectre.

Chaque partiel produit une raie spectrale, et plus précisément un maximum local d'amplitude dans la transformée de Fourier à l'indice correspondant à sa fréquence instantanée. Malheureusement, ce n'est le cas que si la fréquence du partiel est une des fréquences utilisées dans la décomposition de Fourier, ce qui arrive très rarement. La plupart du temps, les indices voisins du maximum local d'amplitude ont également des amplitudes tout à fait significatives (voir figure 2.9). Le nombre de voisins dépend de la fenêtre (w_r) appliquée au signal (voir chapitre « Méthodes d'analyse du signal musical »). La figure 2.10 montre le spectre de puissance des fenêtres de Bartlett (triangulaire) et de Hann (appelée aussi « Hanning »). Ces deux fenêtres sont de bons choix pour la technique *overlap-add* décrite plus loin. Afin de reconstruire la sinusoïde avec une erreur en dessous de -40 dB, douze indices (pic principal et voisins inclus) sont nécessaires dans la transformée de Fourier pour la fenêtre triangulaire, alors que seulement six indices suffisent pour la fenêtre de Hann. Avec douze indices, l'erreur commise en utilisant la fenêtre de Hann est en dessous de -60 dB, ce qui est tout

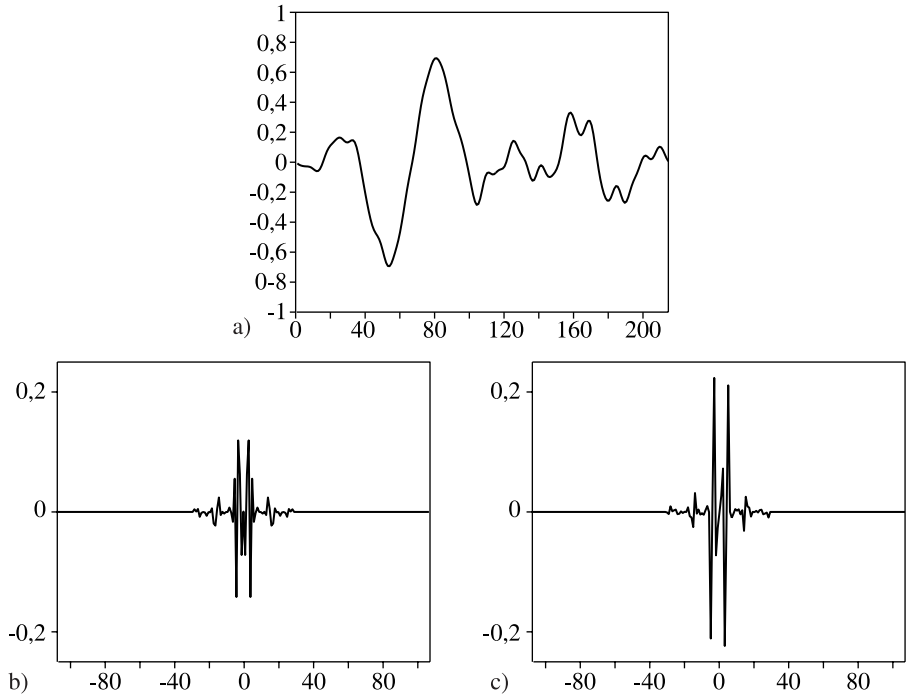


Figure 2.8. *La partie réelle en b (respectivement la partie imaginaire en c) du spectre d'un signal réel (a) est une fonction paire (respectivement impaire)*

à fait acceptable. Il est important de noter qu'en plus du calcul des amplitudes des indices de la transformée de Fourier, le calcul des phases est également nécessaire. Cela peut se faire facilement, en utilisant une technique de fenêtrage dite *zero-phase* utilisée par exemple dans SMS [SERR 97], mais il faut pouvoir retrouver les valeurs des phases des partiels à l'instant t. Seulement la première moitié du spectre doit être reconstruite, puisque l'autre moitié est conjuguée-symmétrique étant donné que le signal sonore attendu après la transformée de Fourier doit être composé uniquement de nombres réels.

2.2.2.3. *Transformée de Fourier rapide inverse*

Une fois l'approximation du spectre à court terme calculée, la transformée de Fourier rapide inverse (IFFT) peut alors être appliquée à ce spectre afin d'obtenir la portion de signal sonore correspondante.

De nombreux algorithmes ont été proposés dans le but d'effectuer la transformée de Fourier et son inverse aussi rapidement que possible. Cooley et Tukey proposent

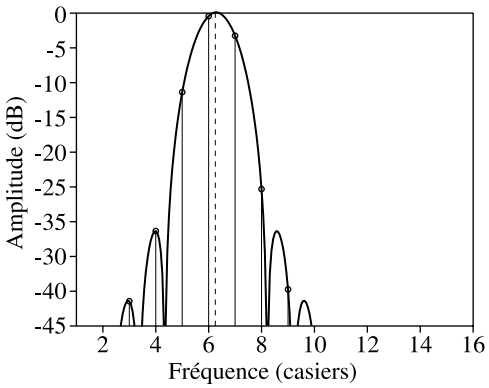


Figure 2.9. *Quand la fréquence du partiel (ici figurée en pointillés) n'est pas un multiple de la plus petite fréquence utilisée dans la transformée de Fourier discrète, les voisins du maximum local d'amplitude ont également des amplitudes tout à fait significatives et dont il faut tenir compte.*

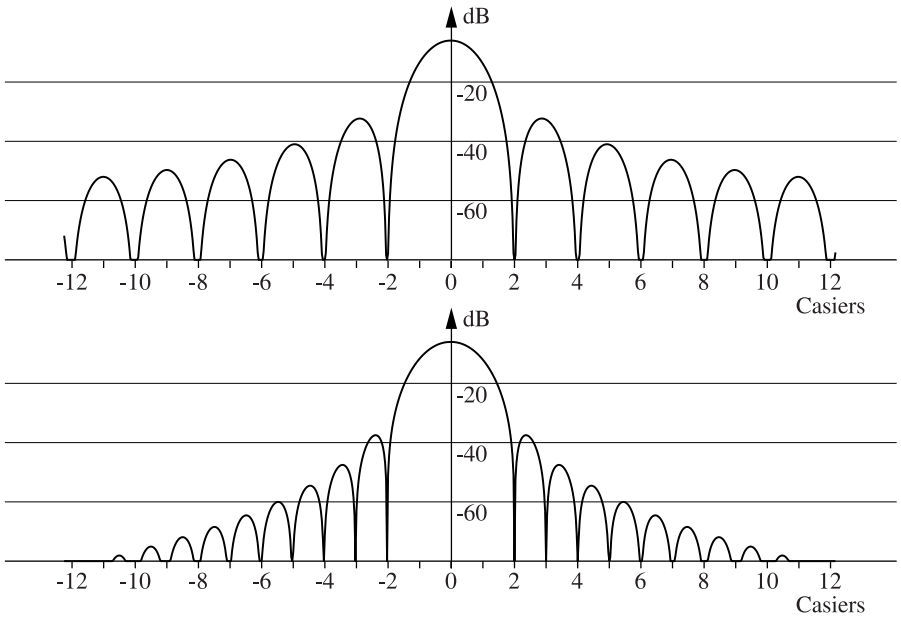


Figure 2.10. *Spectres des fenêtres de Bartlett (en haut) et de Hann (en bas)*

dans [COO 65] le célèbre algorithme *radix-2* (ou *butterfly*) dont une implémentation en langage C est donnée ci-dessous à titre d'information.

```
void
ifft_transform (COMPLEX *spectrum, REAL *samples, int N)
{
  int i, j, d, k;

  ...

  for (d=1; d<N; d+=d)
    {
      for (i=(N/2), j=0; i>0; )
        {
          butterfly (spectrum, j, d, N);
          i--;
          for (k=1; k<d; k++)
            {
              butterfly (spectrum, j+k, d, N);
              i--;
            }
          j += 2*d;
        }
    }

  ...

}

static void
butterfly (COMPLEX *spectrum, int i, int d, int N)
{
  double x;
  COMPLEX w, tmp;

  x = (double) ((i%d)*(N/(2*d)));
  w.re = cos ( (2*M_PI*x) / N );
  w.im = sin ( (2*M_PI*x) / N );

  tmp.re = (spectrum[i+d].re * w.re) - (spectrum[i+d].im * w.im);
  tmp.im = (spectrum[i+d].re * w.im) + (spectrum[i+d].im * w.re);
  spectrum[i+d].re = spectrum[i].re - tmp.re;
  spectrum[i+d].im = spectrum[i].im - tmp.im;
  spectrum[i].re = spectrum[i].re + tmp.re;
  spectrum[i].im = spectrum[i].im + tmp.im;
}
```

Cet algorithme est vraiment très rapide. Si N est la taille de la transformée de Fourier (N étant une puissance de 2), alors la complexité de l'algorithme est en

$O(N \log_2(N))$. Plus précisément, les nombres d'affectations, additions et multiplications sont proportionnels à $N \log_2(N)$. Notons au passage que cet algorithme nécessite le calcul de $N \log_2(N)$ fonctions sinus (ou cosinus), coûteux, même si ces calculs pourraient être réalisés en utilisant, par exemple, l'algorithme efficace du résonateur numérique vu précédemment.

De plus, à la fin de l'algorithme de transformée de Fourier inverse, le signal est stocké sous la forme de nombres complexes, de telle sorte que N affectations sont encore nécessaires pour extraire les parties réelles des échantillons afin de reconstruire le signal sonore réel attendu. Une autre particularité célèbre de cet algorithme est que les nombres complexes se retrouvent stockés à la fin dans un ordre dit *bit-reversed*. Pour retrouver leurs vraies positions dans le vecteur d'échantillons, la représentation binaire de leurs indices doit être inversée avec l'opération de miroir suivante : $k = k_0 k_1 \ldots k_{N-1} \rightarrow k_{N-1} \ldots k_1 k_0 = k'$. Ceci nécessite encore du temps de calcul, raisonnablement faible cependant.

2.2.2.4. *Technique* overlap-add

La transformée de Fourier inverse permet de reconstruire des petits morceaux de signal sonore composés de N échantillons. Afin d'éviter les phénomènes de clic aux frontières de ces différents segments sonores, la technique OLA (*overlap-add*) est avantageusement utilisée ([PEE 98]). Elle consiste en la décomposition du signal sonore en de petites trames temporelles qui se chevauchent dans le temps, par exemple par paquets de $N/2$ échantillons.

Chaque trame temporelle de N échantillons est multipliée par une fenêtre de pondération. La somme des coefficients de toutes les fenêtres superposées doit à tout moment valoir 1. Par conséquent, la fenêtre de pondération qui vient immédiatement à l'esprit est la fenêtre de Bartlett (triangulaire), puisqu'elle vérifie évidemment cette condition (voir figure 2.13). La figure 2.11 illustre la technique *overlap-add*. La trame de signal s est obtenue à partir de son spectre à court terme S en utilisant la FFT inverse (IFFT). Puis s est multipliée par une fonction de pondération w. Si la fenêtre de reconstruction w_r est différente de la fenêtre de recouvrement w_{OLA}, il suffit de prendre $w = w_r / w_{\text{OLA}}$. Finalement, la trame pondérée est additionnée au résultat et le même algorithme est répété $N/2$ échantillons plus loin, et ainsi de suite. Afin d'éviter cette multiplication par w pour chaque trame, une idée astucieuse consiste à utiliser la même fenêtre pour la reconstruction des pics dans le spectre à court terme avant la transformée de Fourier inverse, puisqu'alors la trame de signal résultant de cette transformée sera déjà automatiquement pondérée par la fenêtre de recouvrement adéquate.

Nous avons vu précédemment que la fenêtre de Bartlett (triangulaire) n'est pas très intéressante, puisqu'elle nécessite le calcul d'un grand nombre de points de la transformée de Fourier pour chaque partiel. Du fait de la formule trigonométrique

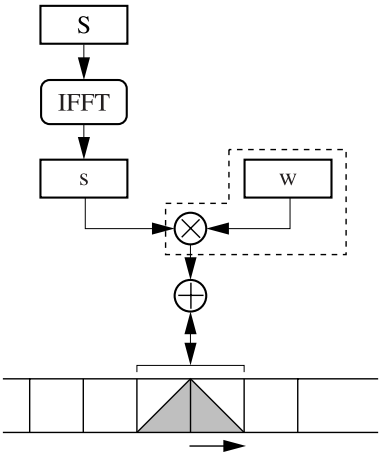


Figure 2.11. *La technique OLA (*overlap-add*). Chaque trame de signal sonore s est obtenue à partir de son spectre à court terme S par le biais d'une FFT inverse (IFFT). Puis s est multipliée par une fenêtre de pondération si nécessaire. Finalement, la trame pondérée est ajoutée au résultat et le même algorithme est répété $N/2$ échantillons plus tard, et ainsi de suite.*

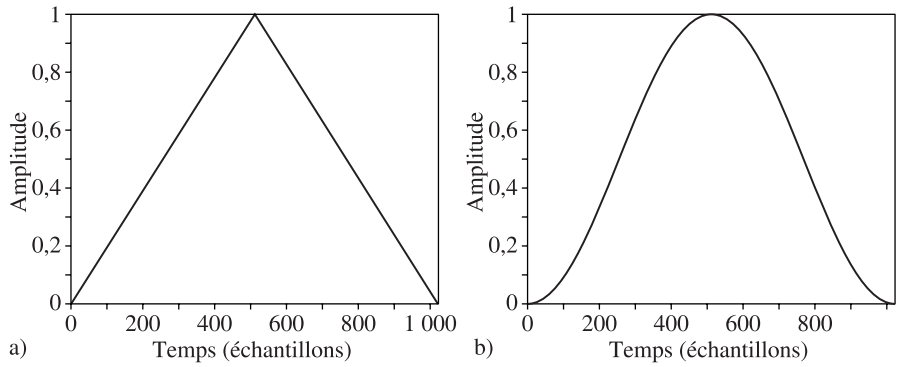


Figure 2.12. *Représentations temporelles des fenêtres de Bartlett (a) et de Hann (b)*

$\sin^2(u)+\cos^2(u) = 1$ bien connue, une autre (bien meilleure) possibilité est la fenêtre en $\sin^2$:

$$w_{\sin^2,N}(n) = \left(\sin\left(\frac{2\pi n}{N}\right)\right)^2 \qquad [2.22]$$

Cependant, considérons la fenêtre de Hann :

$$w_{\text{Hann},N}(n) = \frac{1}{2}\Big(1 - \cos\Big(\frac{2\pi n}{N-1}\Big)\Big) \qquad [2.23]$$

Si N est impair, alors $N/2 = (N-1)/2$ (puisque « / » représente la division euclidienne sur les entiers). Si N est pair, on remplacera $N-1$ par N dans l'équation [2.23], ce qui donnera une fenêtre très proche de la fenêtre de Hann originale. Mais cette fois, nous aurons dans tous les cas :

$$\begin{aligned} & w_{\text{Hann},N}(n) + w_{\text{Hann},N}(n+N/2) \\ & \quad = \frac{1}{2}\Big[1 - \cos\Big(\frac{2\pi n}{N-1}\Big)\Big] + \frac{1}{2}\Big[1 - \cos\Big(\frac{2\pi(n+(N-1)/2)}{N-1}\Big)\Big] \\ & ... = \frac{1}{2}\Big[1 - \cos\Big(\frac{2\pi n}{N-1}\Big)\Big] + \frac{1}{2}\Big[1 - \cos\Big(\frac{2\pi n}{N-1} + \pi\Big)\Big] \\ & ... = \frac{1}{2}\Big[1 - \cos\Big(\frac{2\pi n}{N-1}\Big)\Big] + \frac{1}{2}\Big[1 + \cos\Big(\frac{2\pi n}{N-1}\Big)\Big] \\ & ... = 1 \end{aligned}$$

Par conséquent, cette fenêtre satisfait la condition que la somme de tous les poids des différentes fenêtres superposées doit être égal à 1 (voir figure 2.13). Nous proposons alors d'utiliser la fenêtre de Hann pour la technique *overlap-add*. Comme vu précédemment, c'est en effet un bien meilleur candidat pour la reconstruction des pics dans le spectre. De plus, c'est également une excellente fenêtre de recouvrement.

2.2.3. *Résultats comparatifs*

Nous avons vu précédemment que des oscillateurs logiciels ou la technique FFT $^{-1}$ pouvaient aussi bien être utilisés pour effectuer la synthèse additive. Afin de savoir quelle méthode il vaut mieux employer pour une implémentation temps réel, nous proposons de comparer ici les performances respectives du résonateur numérique (le plus rapide des oscillateurs logiciels) et de la technique FFT $^{-1}$. Deux considérations entrent en jeu : la qualité et la complexité.

2.2.3.1. *Qualité*

Le résonateur numérique permet un contrôle très fin des paramètres de chaque oscillateur, puisque l'algorithme de base peut être modifié pour permettre les variations des paramètres additifs (voir section 2.3). En revanche, la technique FFT $^{-1}$ ne fonctionne que lorsque les paramètres des partiels varient extrêmement lentement, et plus précisément lorsque ces paramètres peuvent être considérés constants pendant les N échantillons de la transformée de Fourier. Quand ces paramètres varient dans le temps, alors le spectre subit une distorsion, étudiée par Masri [MAS 95, MAS 98],

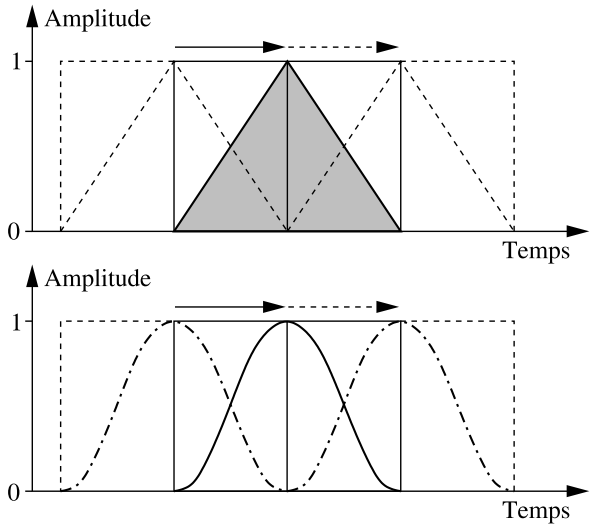


Figure 2.13. *Technique* overlap-add *avec les fenêtres de Bartlett (en haut) ou de Hann (en bas)*

qu'il serait trop coûteux de prendre en considération en termes de temps de calcul. Par conséquent, le résonateur numérique est bien plus flexible et semble être un meilleur choix du point de vue de la qualité.

2.2.3.2. *Complexité*

En première approximation, nous considérerons ici que les différentes instructions (affectation, addition, multiplication) nécessitent à peu près le même temps d'exécution.

Du point de vue de la complexité, le résonateur numérique nécessite, pour chaque partiel, une multiplication et une addition par échantillon, ce qui fait au total $2P$ instructions pour P partiels. Cette technique de synthèse est très efficace. Nous avons mesuré que cet algorithme est capable de synthétiser, en qualité CD (soit une fréquence d'échantillonnage de $F_e = 44\,100$ Hz) et en temps réel, environ deux partiels par MHz de fréquence d'horloge du processeur sur un Intel Pentium II (voir [MAR 99b, MAR 00b, STR 99]), ce qui donne 600 partiels pour un processeur cadencé à 300 MHz. Il est possible de produire encore plus de partiels en tenant compte des caractéristiques des processeurs actuels et plus particulièrement les mécanismes de *pipeline*, sans parler des architectures SIMD (*single instruction, multiple data*) qui permettent d'effectuer la même opération sur des données différentes en parallèle...

Il existe actuellement des processeurs dont la fréquence interne dépasse le gigahertz (soit 1 000 MHz), ce qui permet de synthétiser des milliers de partiels en temps réel. Nous garderons cependant l'exemple du processeur cadencé à 300 MHz pour les comparaisons ultérieures.

Pour obtenir une qualité suffisante avec la technique FFT^{-1}, nous avons vu précédemment que le calcul d'au moins douze points du spectre était nécessaire pour chaque partiel. Alors, en négligeant les affectations nécessaires pour la transformation des complexes en réels et l'algorithme de correction des indices inversés, la technique par transformée de Fourier rapide inverse nécessite $kN\log_2(N)$ instructions. Or, du fait de la technique *overlap-add* elle-même, ce calcul est réalisé deux fois par échantillon. Par conséquent, une valeur optimiste pour le nombre d'instructions par échantillon pour la technique FFT^{-1} avec P partiels est $2kN\log_2(N)$. Déterminer de manière théorique la valeur exacte de k est extrêmement difficile à cause des caractéristiques inhérentes aux processeurs actuels, comme les divers caches et *pipelines*. Toutefois, en pratique, avec l'implémentation très efficace FFTW (*fastest Fourier transform in the West*) du MIT [FRI 98], nous avons mesuré une valeur de $k = 30$ sur un processeur Intel Pentium II. Pour ces mesures de performance, nous avons utilisé FFTW (version 2.1.3), bien que l'algorithme *radix-2* donné plus haut était pratiquement aussi rapide dans ce cas. Plus précisément, avec un processeur cadencé à 300 MHz, la transformée de Fourier inverse prend $2/5$ du temps disponible pour $N = 256$ et ce rapport est proportionnel à $\log_2(N)$. Dans le même temps, le résonateur numérique aurait eu le temps de synthétiser 240 partiels.

Notons que ce temps est utilisé par la technique FFT^{-1} même si un seul partiel doit être synthétisé, ce qui n'est pas le cas avec le résonateur numérique. Bien sûr, quand le nombre de partiels augmente, la technique FFT^{-1} doit être en théorie de plus en plus efficace puisque, pour chaque partiel, le nombre d'instructions nécessaires pour reconstruire son effet dans le spectre à court terme est une constante et n'est pas proportionnel à N, comme c'est le cas pour l'algorithme du résonateur numérique.

Cependant, les choses ne sont pas si simples en pratique, puisque la reconstruction d'un maximum local dans le spectre (et de ces voisins) est coûteuse. Par exemple, les fenêtres rectangulaire ou de Hann nécessitent le calcul de la fonction sinc (sinus cardinal) ; ceci équivaut à un sinus suivi d'une division, ce qui est très coûteux. Bien sûr, d'autres fenêtres peuvent être utilisées, comme la gaussienne tronquée qui, elle, nécessite le calcul de la fonction puissance...

Par conséquent, il apparaît que le résonateur numérique semble aujourd'hui être un meilleur choix. Freed est certainement arrivé à la même conclusion. En effet, bien qu'étant une des personnes à l'origine de la technique FFT^{-1}, il propose avec Hodes [HOD 99] d'utiliser également le résonateur numérique pour la synthèse additive. Cependant, il vaut mieux, pour éviter les éventuels problèmes d'imprécision

numérique, effectuer les opérations arithmétiques en virgule flottante plutôt qu'en virgule fixe.

2.3. Contrôle des partiels

La synthèse additive est contrôlée dans le temps par un flot de paramètres additifs qui varient lentement. Plus exactement, la valeur de chaque paramètre additif (fréquence ou amplitude d'un partiel) est envoyée au processus de synthèse à intervalles de temps réguliers.

2.3.1. *Variation des paramètres*

Le temps T entre deux envois de paramètres est bien plus grand que le temps séparant deux échantillons du signal résultant (la période d'échantillonnage T_e). L'évolution de chaque paramètre additif est alors exprimée sous la forme d'une interpolation entre les valeurs du flot de valeurs discrètes associé. Pour des raisons d'efficacité, T est le plus grand possible afin de réduire au maximum le débit du flot de données contrôlant la synthèse. Dans la section 2.1, nous avons vu que nous pouvions considérer chaque paramètre additif comme un signal (de contrôle) échantillonnable à une fréquence supérieure à $2F_{\max}$. Il suffit donc de prendre $T < 1/(2F_{\max})$.

Mais cela ne signifie pas pour autant que l'on puisse se contenter de changer toutes les T secondes les fréquences et les amplitudes des oscillateurs générant les partiels. En théorie, la fréquence et l'amplitude de chaque partiel peuvent varier légèrement entre chaque nouvel échantillon de l'oscillateur associé au partiel. Idéalement, il faudrait donc mettre à jour les paramètres des oscillateurs tous les échantillons, c'est-à-dire toutes les T_e secondes (T_e étant très inférieure à T), ce qui ferait s'écrouler les performances du processus de synthèse.

Fort heureusement, des considérations psycho-acoustiques font que l'on peut se contenter de changer les paramètres des oscillateurs au minimum tous les 100 échantillons, et ce sans différence audible. Des tests indiquent en effet que changer brutalement l'amplitude ou la fréquence d'un oscillateur n'affecte pas la qualité du son perçu si ce changement est peu important. En revanche, un changement brutal de la phase aurait un effet désastreux : la production de clics audibles. Mais tant que la continuité de la phase est respectée, la fréquence et l'amplitude peuvent être changées brutalement dans des proportions relativement importantes.

En pratique, pour $F_e = 44\,100$ Hz, d'excellents résultats sont obtenus en envoyant les paramètres de contrôle tous les 256 échantillons tout en changeant effectivement les paramètres des oscillateurs tous les 64 échantillons. La figure 2.14 illustre la variation de l'amplitude d'un oscillateur au cours du temps.

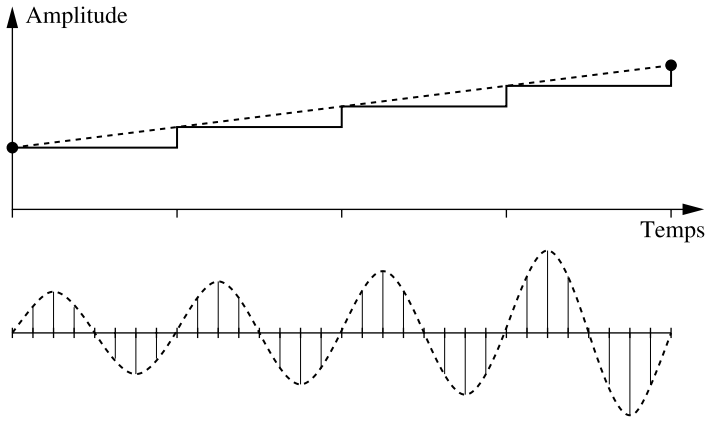


Figure 2.14. *Variation de l'amplitude d'un oscillateur et amplitude du signal audio résultant. Entre deux changements du paramètre amplitude dans le flot de données, quelques valeurs intermédiaires sont calculées par interpolation (quatre dans cet exemple), et, entre deux valeurs interpolées, de nombreux échantillons sont calculés par l'oscillateur.*

2.3.1.1. *Changer les paramètres de contrôle*

Dans les algorithmes de synthèse décrits dans la section 2.2, les paramètres additifs étaient constants, alors qu'ils doivent en fait varier dans le temps. Par conséquent, même si ce n'est que tous les 64 échantillons, il faut être capable de changer l'amplitude et la fréquence de chaque oscillateur pour que le processus de synthèse soit utilisable en pratique. Etant donné la fréquence peu élevée de ces changements, même s'ils sont coûteux en temps machine, leur impact sur les performances du processus de synthèse est négligeable.

Dans le cas de la synthèse par transformée de Fourier inverse, les paramètres ne peuvent être changés qu'entre deux transformées, et donc peu fréquemment. La synthèse par oscillateurs logiciels est bien plus souple. Pour changer l'amplitude, une multiplication par le rapport entre la nouvelle amplitude et l'ancienne suffit. En ce qui concerne le changement de fréquence, pour des algorithmes comme la forme couplée, il suffit de modifier la valeur de l'incrément de phase Δ_ϕ, mais pour le résonateur numérique, les choses sont un peu plus compliquées.

Le résonateur numérique, vu en section 2.2, permet de calculer incrémentalement et de manière optimale, pour le signal discret s, la séquence d'échantillons :

$$s[n] = a\cos(2\pi f T_e n + \phi_0)$$

pour chaque n, où a est l'amplitude, f la fréquence, ϕ_0 la phase initiale et $T_e = 1/F_e$ la période d'échantillonnage en secondes. Nous avons vu précédemment que la valeur

de l'échantillon $s[n]$ pouvait être exprimée en fonction des deux échantillons précédents $s[n-1]$ et $s[n-2]$ ainsi :

$$s[n] = 2\cos(2\pi f/F_e) \cdot s[n-1] - s[n-2] \qquad [2.24]$$

Il existe des variantes de la formule du résonateur numérique qui prennent en considération des évolutions linéaires (et exponentielles) de l'amplitude [GOR 85, SMI 92]. Il est aussi possible de considérer des variations linéaires pour la fréquence. Mais à notre connaissance, aucune formule ne prend en considération les variations des deux paramètres à la fois. De plus, avec ces formules, seules des variations très simples sont considérées, et certainement pas le cas où les paramètres additifs a et f sont des signaux de contrôle dont les variations sont bornées en fréquence.

Intéressons-nous maintenant à la façon de changer les paramètres des oscillateurs (tous les 64 échantillons). Changer l'amplitude a est trivial. On fait varier sa valeur de a_1 à a_2 par une simple multiplication des deux dernières valeurs de l'oscillateur par a_2/a_1. Changer la fréquence f est aisé, puisqu'il s'agit en fait de recalculer $2\cos(\Delta_\phi) = 2\cos(2\pi f/F_e)$ avec la nouvelle valeur de f. Mais la phase ne doit pas être changée car elle doit rester continue.

Etant donné les trois valeurs précédemment calculées par l'oscillateur :

$$\begin{aligned} s[n-1] &= a\cos(\phi - \Delta_\phi) \\ s[n] &= a\cos(\phi) \\ *s[n+1] &= a\cos(\phi + \Delta_\phi) \end{aligned}$$

il est possible de retrouver l'amplitude a et la phase ϕ de la sinusoïde à l'échantillon n. En effet, nous avons :

$$\begin{aligned} & s[n-1] - s[n+1] = 2a\sin(\phi)\sin(\Delta_\phi) \\ & *\big((s[n-1] - s[n+1])/2\big)^2 + \big(\sin(\Delta_\phi)s[n]\big)^2 = a^2\big(\sin(\Delta_\phi)\big)^2 \end{aligned}$$

et finalement :

$$a = \sqrt{\left(\frac{s[n-1] - s[n+1]}{2\sin(\Delta_\phi)}\right)^2 + (s[n])^2} \qquad [2.25]$$

Puisque $\Delta_\phi = 2\pi f/F_e$ avec $0 < f < F_e/2$, nous avons $0 < \Delta_\phi < \pi$ et $\sin(\Delta_\phi) \neq 0$. Par conséquent, la phase est donnée par :

$$\phi = \arccos\Big(\frac{s[n]}{a}\Big) \qquad [2.26]$$

En conclusion, il est possible de changer les paramètres de chaque oscillateur en utilisant l'algorithme suivant (en se référant au système d'équations [2.15]) :

1) retrouver l'amplitude à l'aide de l'équation [2.25]. Bien sûr, il est aussi possible de stocker la valeur courante de l'amplitude du partiel pour éviter de la recalculer, à condition toutefois qu'aucune dérive ne se soit produite entre-temps (à cause d'une éventuelle instabilité numérique des calculs) ;

2) soit simplement multiplier les deux dernières valeurs de l'oscillateur par le rapport entre la nouvelle amplitude et l'ancienne, ce qui assure la continuité de la phase si et seulement si la fréquence n'a pas varié, soit retrouver la phase à partir des derniers échantillons à l'aide de l'équation [2.26] et recalculer les deux valeurs initiales du résonateur à partir de cette phase, de la nouvelle amplitude et de la nouvelle fréquence ;

3) recalculer $\Delta_\phi = \frac{2\pi f}{F_e}$ et $C = 2\cos(\Delta_\phi)$ si la fréquence f a changé.

2.3.1.2. *Eviter les discontinuités*

Il est donc possible d'ajuster l'amplitude et la fréquence de chaque oscillateur à n'importe quel moment, sans introduire de distorsion audible si la variation se fait dans des proportions raisonnables. Mais il est possible de faire encore mieux en termes de qualité.

2.3.1.2.1. Amplitude

La probabilité d'entendre un changement brutal d'amplitude est plus grande quand la valeur absolue de l'échantillon courant est grande. Ceci provient du fait que la différence en amplitude est multipliée par la valeur courante du signal :

$$a_1 \cos(\phi) \rightarrow a_2 \cos(\phi)$$

Plus la valeur (absolue) courante du signal est faible, plus la différence d'amplitude est potentiellement audible lors du changement. Ce phénomène est illustré en figure 2.15. Il est par conséquent souhaitable de changer l'amplitude au bon moment pour garantir la continuité du signal. Le meilleur moment pour changer brutalement la valeur de l'amplitude a de a_1 à a_2 est quand $\cos(\phi)$ est proche de zéro. Plus précisément, les cas les meilleurs et les pires sont respectivement :

– meilleur cas : $\phi \equiv \frac{\pi}{2}\,(\pi)$,

– pire cas : $\phi \equiv 0\,(\pi)$.

2.3.1.2.2. Fréquence

La fréquence présente le problème inverse :

$$\frac{d\phi}{dt} = 2\pi f_1 \rightarrow 2\pi f_2$$

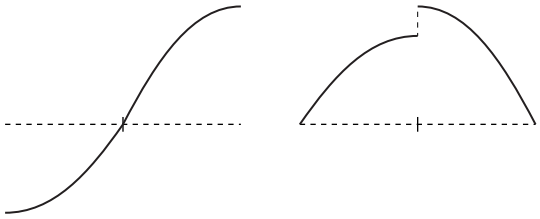


Figure 2.15. *Changement d'amplitude lorsque la valeur absolue du signal est soit minimale (à gauche), soit maximale (à droite). Il apparaît clairement que le cas de gauche est bien meilleur, puisqu'il évite les discontinuités du signal (et donc les clics)*

La probabilité d'entendre un changement brutal de la fréquence est plus grande quand la valeur absolue de l'échantillon courant est faible. Ceci provient du fait que la dérivée du signal change quand la fréquence change et que cette dérivée vaut zéro quand la valeur absolue du signal est la plus grande (maximum local). Ce phénomène est illustré en figure 2.16. Il est donc également souhaitable de changer la fréquence au bon moment pour garantir la continuité de la première dérivée du signal. Etant donné que :

$$\frac{d(a\cos(\phi))}{dt} = -a\frac{d\phi}{dt}\sin(\phi)$$

le meilleur moment pour changer brutalement la valeur de la fréquence f de f_1 à f_2 est quand $\sin(\phi)$ est proche de zéro. Plus précisément, les cas les meilleurs et les pires sont respectivement :

– meilleur cas : $\phi \equiv 0\,(\pi)$,

– pire cas : $\phi \equiv \frac{\pi}{2}\,(\pi)$.

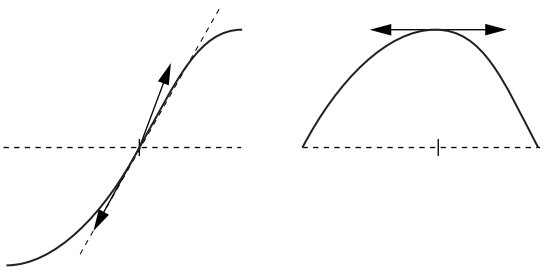


Figure 2.16. *Changement de fréquence lorsque la valeur absolue du signal est soit minimale (à gauche), soit maximale (à droite). Il apparaît clairement que le cas de droite est bien meilleur, puisqu'il évite les discontinuités de la dérivée du signal (qui produisent également des clics).*

Un problème survient alors avec les basses fréquences, car les instants idéaux pour mettre à jour les paramètres sont trop espacés dans le temps. Par exemple, un oscillateur de fréquence $f = 50\,\text{Hz}$ échantillonné à $F_e = 44\,100\,\text{Hz}$ a des instants de mise à jour qui se produisent tous les 441 échantillons, ce qui est bien plus que la période de mise à jour maximale de 64 échantillons décidée auparavant pour des raisons psycho-acoustiques. Dans ce cas, on peut renoncer à une mise à jour optimale des paramètres. D'autres possibilités et une étude détaillée des façons de prendre en considération efficacement les variations des paramètres additifs lors de la synthèse peuvent être trouvées dans [MAR 00b].

2.3.2. *Rééchantillonnage par les splines*

Les paramètres additifs sont des fonctions du temps qui varient lentement. Plus précisément, ces fonctions peuvent être considérées comme des signaux (de contrôle) dont les variations sont bornées en fréquence. La fréquence maximale $F_{\max}$ est inférieure à la plus petite fréquence audible, aux alentours de 20 Hz. Connaissant cela, nous n'allons pas nous contenter d'une simple interpolation linéaire pour retrouver à chaque instant la valeur d'un des paramètres. Il est en effet possible de reconstruire les variations des paramètres à partir de leurs échantillons discrets, et ce avec une grande précision.

2.3.2.1. *Rééchantillonnage classique*

Si, par exemple, les paramètres de contrôle sont envoyés en entrée du processus de synthèse seulement tous les 256 échantillons du signal de sortie et si les paramètres de chaque oscillateur doivent être mis à jour tous les 64 échantillons, alors les signaux de contrôle discrets doivent être suréchantillonnés avec un facteur 4 ($256/64$).

Ce rééchantillonnage peut être réalisé à l'aide d'un filtre de reconstruction classique dont la réponse impulsionnelle r est une fonction sinus cardinal multipliée par une fenêtre en cloche w comme la fenêtre de Hann par exemple. Cette technique est expliquée en détail dans [SMI 84, SMI 00]. Plus formellement, nous avons :

$$r(t) = w(t)\text{sinc}(F_{\max}\, t) \qquad [2.27]$$

avec :

$$\text{sinc}(t) = \frac{\sin(\pi t)}{\pi t} \quad (\text{et } \text{sinc}(0) = 1) \qquad [2.28]$$

où $F_{\max}$ est la borne supérieure en fréquence des variations des paramètres additifs, aux alentours de 20 Hz donc. La fenêtre de Hann discrète de taille N est définie par l'équation suivante :

$$w_{\text{Hann},N}[n] = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right) \qquad [2.29]$$

Mais ce procédé de rééchantillonnage classique nécessite une convolution qui dégrade sérieusement les performances de l'algorithme de synthèse. Le problème est donc de trouver efficacement les valeurs intermédiaires des paramètres additifs, sans utiliser de filtre de reconstruction. Il est possible d'utiliser une approximation de cette reconstruction idéale.

2.3.2.2. *Splines cardinales*

Le principe des splines est d'utiliser des polynômes afin d'approximer ou d'interpoler des fonctions. Nous nous intéresserons ici aux splines d'interpolation pour la reconstruction uniforme. Les splines cardinales sont des splines uniformes d'interpolation dont l'usage est très répandu en informatique graphique [FOL 95]. Tout signal s continu peut être reconstruit à partir de ses échantillons (discrets) en utilisant l'équation suivante :

$$s(pT+t) = \sum_{i=0}^{3} c_i(t)s[p-1+i] \quad (0 \leqslant t < 1) \qquad [2.30]$$

où T est la période d'échantillonnage du signal (de contrôle) s et les fonctions c_i sont, pour les splines cubiques, les fonctions de mélange de Hermite suivantes :

$$c_0(t) = \frac{1}{2}(-t + 2t^2 - t^3) \qquad [2.31]$$

$$c_1(t) = \frac{1}{2}(2 - 5t^2 + 3t^3) \qquad [2.32]$$

$$c_2(t) = \frac{1}{2}(t + 4t^2 - 3t^3) \qquad [2.33]$$

$$c_3(t) = \frac{1}{2}(-t^2 + t^3) \qquad [2.34]$$

Ces quatre fonctions polynomiales sont représentées sur la figure 2.17. Etant donné que la célèbre méthode de Horner peut être utilisée pour réaliser le calcul des polynômes, le processus de reconstruction est très efficace en termes de temps de calcul. La reconstruction se révèle être également de très bonne qualité, car les fonctions de Hermite constituent en fait une approximation polynomiale par morceaux de la réponse impulsionnelle d'un filtre de reconstruction, l'équation [2.30] étant alors une petite convolution entre le signal discret et cette approximation. Plus précisément, la comparaison entre les fonctions de Hermite et la fonction sinus cardinal (tronquée) à la base des filtres de reconstruction est illustrée en figure 2.18. Les comportements fréquentiels sont également très semblables. Dans les deux cas, on est en présence d'un filtre passe bas, avec une fréquence de coupure aux alentours de $F_{\max}$. Même si les splines cardinales constituent un filtre de qualité moindre en comparaison avec une fonction sinus cardinal fenêtrée, ces splines sont une bonne approximation de la reconstruction idéale et permettent de gagner beaucoup de temps lors des calculs.

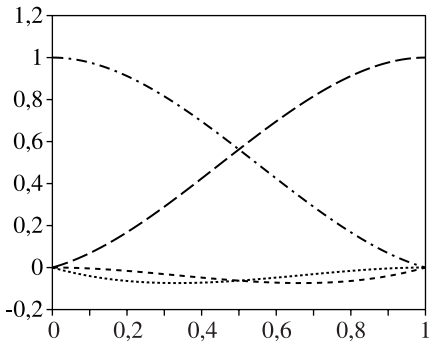


Figure 2.17. *Fonctions de mélange de Hermite pour les splines cardinales cubiques*

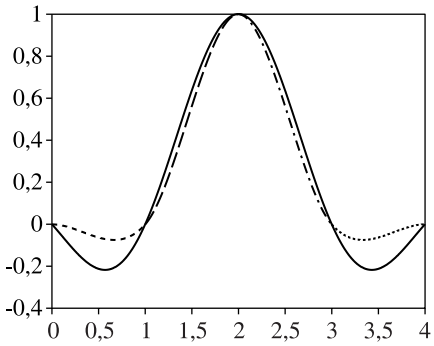


Figure 2.18. *Les fonctions de Hermite (réordonnées dans le temps) comparées à la fonction sinus cardinal*

2.4. Accélération par la psycho-acoustique

Avec les méthodes de synthèse vues précédemment, il est actuellement possible de générer en temps réel des centaines, voire des milliers, d'oscillations quasi sinusoïdales.

Les fluctuations rapides de la pression de l'air au niveau des oreilles génèrent une sensation auditive. Il se trouve que le mot *son* désigne aussi bien la vibration physique que la sensation qu'elle procure. De plus amples informations sur la psycho-acoustique peuvent être trouvées dans [KIT 94, ZWI 81, ZWI 90, ZWI 91]. Comme le dit si bien Risset [RIS 88] : « Avec l'ordinateur, on peut bâtir un son de structure physique arbitraire : mais ce qui importe, c'est son effet sensible. »

Un son harmonique avec une fréquence fondamentale de 50 Hz comporte au plus 440 partiels (puisque la plus grande fréquence audible considérée est de 22 kHz). En fait, ce nombre est suffisant pour la plupart des sons possibles, même polyphoniques. En effet, quand le nombre de partiels est grand, les volumes de nombreux partiels sont très faibles et la distance moyenne en fréquence entre deux partiels consécutifs est très réduite. Des considérations psycho-acoustiques montrent alors que beaucoup de ces partiels sont inaudibles. Plus précisément, en prenant en considération des phénomènes psycho-acoustiques comme le seuil d'audibilité et le masquage, il est possible, en ignorant ces partiels inaudibles, d'économiser du temps de calcul, et ainsi d'accélérer grandement le processus de synthèse.

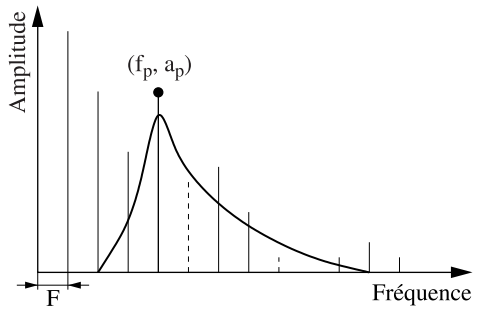


Figure 2.19. *Partiels masqués dans un son harmonique. Les deux partiels sous le seuil de masquage peuvent être ignorés par le processus de synthèse, puisqu'ils sont inaudibles*

2.4.1. *Echelles perceptives*

Si, à l'instant t, nous pouvons mesurer l'amplitude et la fréquence instantanées d'un son (ou d'un de ses partiels), les paramètres perceptifs correspondants sont respectivement l'intensité et la hauteur. La loi de Fechner, qui s'applique à tout organe sensoriel, affirme que la sensation est proportionnelle au logarithme de l'excitation. Par conséquent, les êtres humains perçoivent ces paramètres sur des échelles logarithmiques.

2.4.1.1. *L'échelle des décibels (dB)*

Cette échelle est communément utilisée pour représenter le volume. Les relations entre le volume en dB et l'amplitude linéaire sont données par les équations [2.35] et [2.36]. En considérant que l'amplitude maximale (1,0 dans l'échelle linéaire) doit correspondre à un volume de 120 dB afin de se conformer aux dB SPL (*Sound Pressure Level*) standard, il faut prendre $A_{0\,\mathrm{dB}} = 10^{-6}$, correspondant à une pression acoustique de $P_{0\,\mathrm{dB}} = 2 \times 10^{-5}$ Pa (pascals). De toute façon, l'amplitude et la

pression étant proportionnelles, c'est juste une affaire de translation de l'origine des volumes (0 dB) dans l'échelle logarithmique :

$$V(a) = 20\ \log_{10}\Big(\frac{a}{A_{0\,\mathrm{dB}}}\Big) \qquad [2.35]$$

$$A(v) = A_{0\,\mathrm{dB}}\ 10^{v/(20\,\mathrm{dB})} \qquad [2.36]$$

2.4.1.2. *L'échelle Bark (d'après Barkhausen)*

Cette échelle est très utile pour représenter les fréquences car elle est beaucoup plus proche de notre perception que l'échelle linéaire de Hertz [ZWI 91]. Les équations [2.37] et [2.38] permettent de passer de l'échelle Hertz à l'échelle Bark et réciproquement :

$$B(f) = \begin{cases} f/100 & \text{si } f \leqslant 500 \\ 9 + 4\ \log_2(f/1\,000) & \text{si } f > 500 \end{cases} \qquad [2.37]$$

$$F(b) = \begin{cases} 100\,b & \text{si } b \leqslant 5 \\ 1\,000 \times 2^{(b-9)/4} & \text{si } b > 5 \end{cases} \qquad [2.38]$$

2.4.2. *Seuil d'audibilité*

Les êtres humains peuvent entendre des fréquences dans un intervalle allant approximativement de 20 Hz à 22 kHz, mais le seuil de sensibilité en amplitude $\mathcal{S}_a$ dépend de la fréquence. L'équation [2.39] donne une bonne approximation de ce seuil. Les partiels dont les volumes sont inférieurs au seuil ne sont pas audibles et peuvent par conséquent être ignorés durant la synthèse :

$$\mathcal{S}_a(f) = 3{,}64\,(f/1\,000)^{-0{,}8} - 6{,}5\,e^{-0{,}6(f/1\,000-3{,}3)^2} + 10^{-3}\,(f/1\,000)^4 \qquad [2.39]$$

2.4.3. *Phénomène de masquage*

D'un point de vue strictement physique, l'addition de deux signaux de même amplitude est régie par une loi d'addition non linéaire et donne un maximum de $+6$ dB (dans le cas de deux signaux identiques en phase). Toutefois, d'un point de vue perceptif, il y a modification du seuil de perception pour un son m (son masqué) lorsqu'il est joué en même temps qu'un son plus fort M (son masquant). Ce phénomène est connu sous le nom de masquage fréquentiel [WEG 24, ZWI 81, ZWI 90, ZWI 91]. Considérons le cas où M et m sont deux sinusoïdes de fréquences respectives

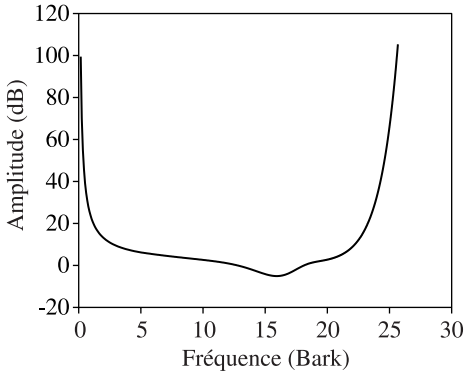


Figure 2.20. *Seuil d'audibilité* $\mathcal{S}_a$

f_M et f_m et d'amplitudes respectives a_M et a_m. Supposons de plus que $a_M > a_m$. Si f_m est proche de f_M, le son m est masqué par le son M et devient inaudible. Ce phénomène peut être avantageusement utilisé pour réduire le nombre d'oscillations sinusoïdales à générer lors du processus de synthèse additive, en ignorant les partiels masqués (inaudibles).

Garcia et Pampin utilisent dans [GAR 99] ce phénomène de masquage pour réduire le nombre de partiels des sons dans le modèle additif. Le phénomène de masquage est également utilisé dans la compression audio MPEG niveau 3 (MP3) [BRA 95, HER 95, JAY 93, PAI 01, PAN 95, PAP 95]. Il est tout à fait possible d'utiliser une technique similaire pour l'accélération de la synthèse sonore plutôt qu'à des fins de compression du son. Nous ne souhaitons en aucun cas compresser les sons, car il nous faut garder tous les partiels – mêmes inaudibles – pour les transformations musicales. Mais les partiels masqués peuvent être ignorés lors de la synthèse car ils sont inaudibles. En première approximation, nous pouvons considérer que le seuil de masquage est quasiment un triangle en échelles Bark-dB, comme indiqué sur la figure 2.21. Garcia et Pampin [GAR 99] utilisent un modèle de masquage simple pour évaluer le rapport signal/masque (en anglais, *signal-to-mask ratio*, SMR) de chaque partiel. Ce modèle est caractérisé par :

– la différence Δ entre le volume du partiel masquant et son masque (-10 dB) ;

– le demi-triangle de masquage vers les fréquences basses (pente gauche : 27 dB/Bark) ;

– le demi-triangle de masquage vers les fréquences élevées (pente droite : -15 dB/Bark).

Bien évidemment, comme toujours avec la perception, la réalité est bien plus complexe. Des battements peuvent être perçus quand $f_m \approx nf_M$ (n désignant un entier

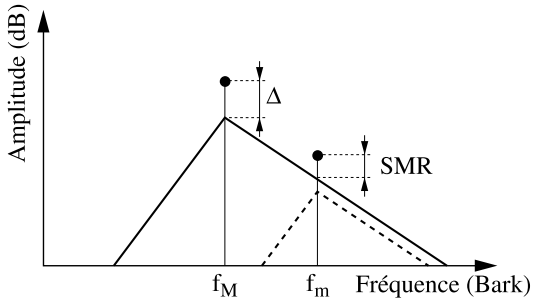


Figure 2.21. *Masquage d'une sinusoïde de fréquence f_m par une autre sinusoïde de fréquence f_M. L'effet de masquage est maximal quand f_m et f_M sont proches. En première approximation, nous pouvons considérer que le seuil de masquage est quasiment un triangle en échelles Bark-dB, même si ce n'est pas exactement le cas en pratique, particulièrement pour le sommet du triangle.*

positif). De plus, les deux sons peuvent interagir pour créer perceptivement des fréquences $i\,f_m \pm j\,f_M$ (i et j étant des entiers), créant ainsi des harmoniques « virtuelles » pour une sinusoïde pure. Cela peut conduire à la perception d'une fréquence fondamentale virtuelle même si cette dernière est physiquement absente du signal sonore.

Un autre phénomène très intéressant est le masquage temporel. Il en existe deux sortes. Le postmasquage temporel survient quand le son masquant disparaît. En fait, l'effet du masquage fréquentiel persiste, en s'estompant, pendant plusieurs millisecondes après cette disparition. Le plus surprenant est qu'il existe aussi un phénomène de prémasquage temporel. Plus précisément, on peut considérer que l'effet de masquage est actif quelques millisecondes avant que le son masquant n'apparaisse vraiment. Toutefois, ce phénomène est beaucoup moins prononcé et peut être ignoré.

2.4.4. *Algorithme général*

Avant l'algorithme de synthèse additive lui-même (voir section 2.2), un autre algorithme peut être ajouté afin de réduire le nombre de partiels à synthétiser, en enlevant les partiels inaudibles.

Cet algorithme est très simple. Il consiste d'abord en la suppression des partiels sous le seuil d'audibilité. Puis, cet algorithme parcourt les partiels ordonnés par amplitudes décroissantes et décide pour chaque partiel s'il est masqué ou non, en calculant

dans le même temps le masque global M, et ce de manière incrémentale. Pour chaque partiel p d'amplitude a_p (de volume $V(a_p)$, voir équation [2.35]) et de fréquence f_p :

– si $M(f_p) + \Delta < V(a_p)$, alors p est un partiel masquant et M doit être mis à jour avec sa contribution ;

– si $M(f_p) < V(a_p) \leqslant M(f_p) + \Delta$, alors p n'est ni masquant ni masqué ;

– si $V(a_p) \leqslant M(f_p)$, alors p est simplement masqué.

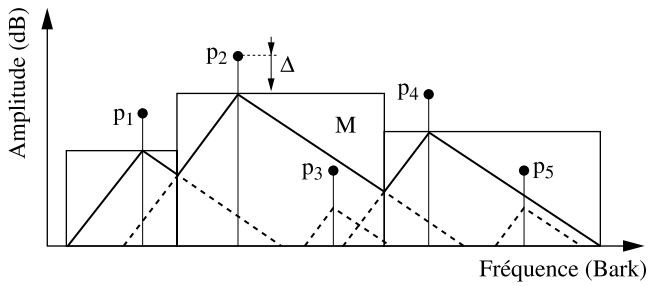


Figure 2.22. *Cinq partiels et le masque associé M (ligne polygonale en gras) ; p_1, p_2 et p_4 sont des partiels masquants et contribuent au masque M. Les zones de fréquence de leurs contributions sont représentées par des rectangles. p_5 n'est ni masquant ni masqué et p_3 est masqué (par p_2).*

Pour construire le masque global M, plusieurs possibilités s'offrent à nous.

2.4.4.1. *Approche discrète*

Une première possibilité, discrète, consiste à en maintenir une version échantillonnée uniformément selon l'échelle Bark. Au début, la valeur de M est positionnée à 0 ($-\infty$ dB). Puis, lorsque l'on rencontre un partiel masquant lors du parcours des partiels, la contribution de ce partiel est accumulée dans le masque global M : pour chaque échantillon de M, on conserve la plus grande valeur entre le triangle de masquage associé au partiel courant et l'ancienne valeur du masque global M. Cet algorithme nécessite un échantillonnage de M suffisamment fin pour obtenir une bonne précision. Mais le nombre d'échantillons doit être relativement peu élevé afin que la complexité de l'algorithme reste acceptable pour une synthèse en temps réel. En effet, il doit rester plus avantageux de supprimer les partiels inutiles (inaudibles) plutôt que de les synthétiser quand même... En pratique, 2 048 échantillons représentent un bon compromis entre la précision et la rapidité.

2.4.4.2. *Approche continue*

Une seconde possibilité est une approche continue. Les contributions des partiels masquants au masque global M peuvent être avantageusement stockées dans une liste

ordonnée selon les fréquences croissantes. Pour décider si un nouveau partiel p est masquant, ni masquant ni masqué ou bien masqué, il suffit d'effectuer une recherche dans cette liste afin de trouver les deux contributions (gauche et droite) entourant la fréquence f_p du nouveau partiel p. S'il apparaît que p est un partiel masquant, alors sa contribution doit être insérée dans la liste entre les contributions voisines (afin de maintenir l'ordre de la liste) et ces dernières doivent être mises à jour. Le partiel courant p ne peut pas les masquer car son amplitude est plus faible, mais il peut réduire les plages de fréquences de leurs contributions au masque global M. Cet algorithme nécessite une structure de données ordonnée en fréquence, avec la notion de voisins gauche et droit, et où la recherche et l'insertion sont aussi rapides que possible. Pugh propose une telle structure de données dans [PUG 90] : la *skip list*. La complexité d'une recherche ou d'une insertion dans une *skip list* de n éléments est en $O(\log(n))$, ce qui est optimal. L'utilisation de cette structure de données pour l'accélération de la synthèse sonore peut être trouvée dans [LAG 01a].

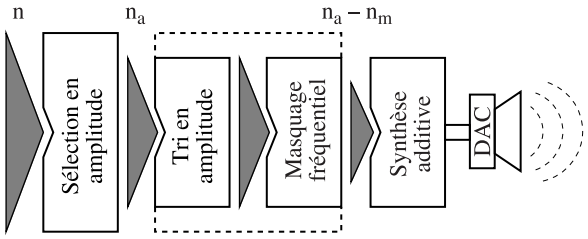


Figure 2.23. *Architecture logicielle générale. Les partiels sous le seuil d'audibilité sont supprimés. Les n_a partiels restants sont ordonnés par amplitudes décroissantes, puis parcourus un à un tout en construisant le masque global M et en éliminant les n_m partiels masqués, c'est-à-dire situés sous ce masque M. Les $n_a - n_m$ partiels restants sont alors synthétisés avec un algorithme de synthèse additive classique (voir section 2.2).*

Il serait souhaitable de considérer, dans le futur, d'autres phénomènes psychoacoustiques comme le masquage temporel. Ceci permettrait de réduire davantage le nombre de partiels à synthétiser et d'augmenter ainsi la performance des algorithmes de synthèse.

2.5. Techniques non linéaires

Jusqu'à présent, les techniques de synthèse étudiées dans ce chapitre nécessitaient un oscillateur pour chaque partiel à générer. Pour ces techniques dites linéaires, l'ensemble des partiels et celui des oscillateurs sont en bijection. Il existe également des techniques non linéaires qui permettent de produire des sons complexes, composés d'un grand nombre de partiels, à l'aide d'un nombre d'oscillateurs réduit.

2.5.1. *Remarque préliminaire*

Considérons les deux signaux sonores s_1 et s_2 suivants :

$$s_1(t) = \cos(2\pi\,5t)\cos\Big(2\pi\,4\,000t - \frac{\pi}{2}\Big) \qquad [2.40]$$

c'est-à-dire l'équation [2.1] avec $P = 1$, $a_1(t) = \cos(2\pi\,5t)$, $f_1(t) = 4\,000$ Hz, $\phi_1 = -\pi/2$ et :

$$s_2(t) = \frac{1}{2}\cos\Big(2\pi\,3\,995t - \frac{\pi}{2}\Big) + \frac{1}{2}\cos\Big(2\pi\,4\,005t - \frac{\pi}{2}\Big) \qquad [2.41]$$

c'est-à-dire l'équation [2.1] avec $P = 2$, $a_1(t) = 1/2$, $f_1(t) = 3\,995$ Hz, $\phi_1 = -\pi/2$, $a_2(t) = 1/2$, $f_2(t) = 4\,005$ Hz, $\phi_2 = -\pi/2$.

En fait, ces deux sons sont les mêmes, $s_1 = s_2$, et ce du fait des équations trigonométriques suivantes :

$$\cos(p) + \cos(q) = 2\cos\Big(\frac{p-q}{2}\Big)\cos\Big(\frac{p+q}{2}\Big) \qquad [2.42]$$

$$\cos(a)\cos(b) = \frac{1}{2}\big(\cos(a-b) + \cos(a+b)\big) \qquad [2.43]$$

D'un point de vue strictement mathématique, les deux formulations sont équivalentes. Mais, pour rester en accord avec la perception, l'équation [2.40] doit être préférée à l'équation [2.41], car on entend réellement une fréquence de 4 000 Hz qui subit un tremolo de 5 Hz. L'oreille décide lorsque les mathématiques ne le peuvent pas. En fait, quand la différence entre p et q dans l'équation [2.42] est inférieure à un seuil aux alentours de 20 Hz (qui est la plus petite fréquence audible), alors il faut privilégier une sinusoïde (audible) multipliée par une autre, de fréquence inaudible. Mais si l'on ne tient pas à conserver la cohérence psycho-acoustique des paramètres du modèle, il est possible d'utiliser la modulation pour générer beaucoup de partiels avec peu d'oscillateurs.

2.5.2. *Synthèse par modulation d'amplitude*

La modulation d'amplitude consiste par exemple à multiplier (moduler) un ensemble de n oscillateurs (son modulé) par un autre oscillateur (oscillateur modulant) afin de créer un timbre encore plus complexe. En effet, on déduit de l'équation [2.43] que :

$$\Big(\sum_{k=1}^{n}\cos(a_k)\Big)\cos(b) = \frac{1}{2}\Big[\sum_{k=1}^{n}\cos(a_k - b) + \sum_{k=1}^{n}\cos(a_k + b)\Big]$$

Il est donc possible de générer $2n$ partiels avec seulement $n + 1$ oscillateurs.

2.5.3. *Synthèse par modulation de fréquence*

Il est encore plus intéressant de composer (au sens mathématique de la composition des fonctions) les oscillateurs entre eux, plutôt que de simplement les multiplier. Chowning, avec la synthèse par modulation de fréquence [CHO 73], a introduit une nouvelle technique de synthèse utilisée dans de nombreux synthétiseurs matériels (voir aussi [MOO 90, MOOR 85]). Il s'agit en fait d'une modulation de la phase d'un oscillateur par le signal produit par un autre oscillateur, comme dans l'équation [2.44] :

$$s(t) = A(t)\sin\big(2\pi f_c(t)t + I(t)\sin(2\pi f_m(t)t)\big) \qquad [2.44]$$

Dans l'équation [2.44], A est l'amplitude, f_c est la fréquence porteuse (*carrier*, en anglais), I est appelé indice de modulation et f_m est la fréquence de modulation. Toutes ces fonctions peuvent varier lentement dans le temps. Pour comprendre comment il est possible de générer des sons musicaux de cette façon, il suffit d'examiner le membre droit de l'équation [2.44] et de le voir sous la forme de somme de sinusoïdes grâce à la formule mathématique [ABR 66] suivante :

$$\sin\big(\theta+a\sin(\beta)\big) = J_0(a)\sin(\theta)+\sum_{k=1}^{\infty} J_k(a)\big(\sin(\theta+k\beta)+(-1)^k\sin(\theta-k\beta)\big) \qquad [2.45]$$

où $J_k(a)$ est la k-ième fonction de Bessel du premier ordre au point a. Ces fonctions de Bessel sont données par l'équation :

$$J_n(x) = \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{n+2k}}{k!\Gamma(n+k+1)} \quad (n \geqslant 0) \qquad [2.46]$$

où :

$$\Gamma(n) = \int_0^{\infty} t^{n-1}e^{-t}\,dt \quad (n > 0) \qquad [2.47]$$

Si n est un entier, on a $\Gamma(n+1) = n!$ (factorielle n). On peut relier l'équation [2.45] à l'équation [2.44] en prenant $\theta = 2\pi f_c t$, $\beta = 2\pi f_m t$ et $a = I$. L'équation [2.45] montre alors qu'il est possible de produire tout un ensemble de partiels distants en fréquence de f_m à l'aide de seulement deux oscillateurs sinusoïdaux. En fait, il est possible de produire un spectre harmonique si f_c/f_m est un nombre rationnel. Par exemple, le choix $f_c = f_m$ produit un spectre harmonique dont les amplitudes des partiels sont des sommes de paires de fonctions de Bessel. En utilisant le fait que $\sin(-x) = -\sin(x)$ dans les équations [2.44] et [2.45], la décomposition en série de Fourier du signal sonore produit dans le cas où $f_c = f_m$ est :

$$\sin\big(\theta + a\sin(\theta)\big) = \sum_{k=1}^{\infty}\Big(J_{k-1}(a) + (-1)^k J_{k+1}(a)\Big)\sin(k\theta) \qquad [2.48]$$

Comme on a $\theta = \beta = 2\pi f_m t$, il est clair que l'équation [2.48] représente une série harmonique. Les partiels (ou harmoniques) sont ordonnés par fréquences croissantes, multiples de la fréquence fondamentale f_m, et leurs amplitudes sont des sommes de fonctions de Bessel. Il est alors possible de contrôler le spectre des sons produits en agissant sur l'indice de modulation $I = a$.

Lorsque cet indice vaut 0, on obtient un sinusoïde pure car $J_0(0) = 1$ et $\forall k > 0$, $J_k(0) = 0$. La figure 2.24 montre quelques exemples de fonctions de Bessel du premier ordre. Quand I augmente, le spectre s'enrichit de nouveaux partiels de part et d'autre de la fréquence centrale f_c. En règle générale, la fréquence porteuse f_c fixe l'origine du spectre et la fréquence modulante f_m détermine l'espacement des partiels en fréquence. En contrôlant l'indice I dans le temps, il est possible d'obtenir des variations du spectre tout à fait intéressantes (grâce aux caractéristiques des fonctions de Bessel). En faisant augmenter I rapidement à partir de 0, on peut obtenir des sons de cuivres assez réalistes, comme des sons de trompettes par exemple.

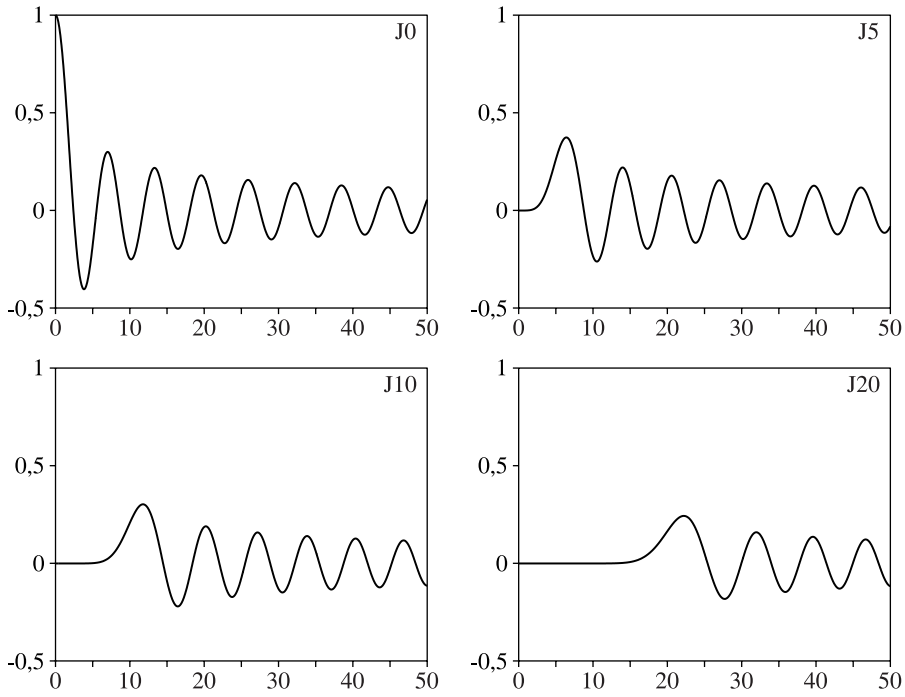


Figure 2.24. *Exemples de fonctions de Bessel du premier ordre*

D'autres relations entre les fréquences f_c et f_m peuvent donner d'autres sons intéressants. Par exemple, les cas où $f_m = 2f_c$ donnent uniquement des harmoniques

impaires, ce qui est très utile pour imiter les sons de clarinettes. En prenant $\beta = 2\theta$, l'équation [2.44] devient en effet :

$$\sin\big(\theta + a\sin(2\theta)\big) = \sum_{k=0}^{\infty}\big(J_k(a) + (-1)^k J_{k+1}(a)\big)\sin\big((2k+1)\theta\big) \qquad [2.49]$$

En prenant $f_m = \sqrt{2}f_c$, on obtient des spectres inharmoniques qui permettent de simuler des gongs ou des cloches. Evidemment, il est possible de combiner plusieurs oscillateurs entre eux, soit en série (cascades de modulations), soit en parallèle (simple superposition). Les synthétiseurs matériels, comme le DX7 de Yamaha, peuvent proposer par exemple trente-deux façons de combiner six oscillateurs. Il est aussi possible d'utiliser la rétroaction (utiliser la sortie d'un oscillateur pour le moduler lui-même) afin de générer des bruits. D'autres exemples très détaillés peuvent être trouvés dans [CHO 73, MOR 77, SCH 77].

Il existe d'autres techniques de synthèse non linéaire, comme par exemple celle proposée par Arfib dans [ARF 79]. Le principal problème avec ces techniques de synthèse est que les paramètres sonores sont souvent très éloignés de la perception et qu'il en résulte une perte de maîtrise lors de l'édition des sons. Cet inconvénient peut aussi se révéler être un avantage pour un créateur souhaitant explorer des timbres nouveaux en jouant avec le hasard. De plus, ces techniques ne comportent pas de méthode d'analyse, ce qui rend quasi impossible la reproduction de sons existants. Les sons produits sont donc purement synthétiques, toutefois les timbres obtenus sont souvent très beaux.

2.6. Synthétiser les bruits

Les physiciens définissent le bruit comme étant un signal indésirable qui interfère avec un autre signal bien précis. La définition musicale du bruit, elle, est loin d'être claire. Nous utiliserons ici le mot *bruit* pour désigner un signal aléatoire résultant d'un processus stochastique. Pour les « signaux déterministes », comme les sommes de sinusoïdes vues précédemment, chaque valeur de la séquence d'échantillons est complètement déterminée, et ce de manière unique, par une expression mathématique ou une règle d'un type bien précis. Mais dans beaucoup de situations, les processus qui génèrent les signaux sont si complexes que la description précise de ces signaux est extrêmement difficile, voire impossible. Il peut alors être indispensable de modéliser ces signaux comme résultats de processus stochastiques. Un « signal stochastique » est considéré comme appartenant à une classe de signaux caractérisés par un ensemble de fonctions de densité de probabilité. En d'autres termes, pour un signal s à un temps t, la valeur $s(t)$ est supposée résulter d'un schéma de probabilités sous-jacent.

2.6.1. *Approximation du spectre*

Le spectre d'un bruit peut être approché par un spectre composé de pics de fréquences très proches [ZWI 81]. Quand la distance entre deux pics adjacents n'excède pas 1 % (10 Hz autour de 1 000 Hz), c'est en fait une excellente approximation. Par conséquent, avec une taille suffisante, la transformée de Fourier inverse peut aussi synthétiser des bruits. Nous nous intéresserons dans la suite uniquement aux bruits modélisés sous la forme de bruit blanc filtré, caractérisés par leur enveloppe spectrale, comme dans le modèle SMS défini précédemment (voir section 2.1).

L'idée est de fixer la valeur de la composante continue (composante DC, indice 0 du spectre discret) à 0, puis d'utiliser les amplitudes données par une enveloppe spectrale pour les amplitudes des autres indices (composantes AC). Les phases sont tirées au hasard, et ce pour chaque spectre à court terme. En fait, seule la première moitié du spectre doit être reconstruite, puisqu'une condition nécessaire et suffisante pour que le signal résultant soit réel est que l'autre moitié du spectre soit conjuguée-symétrique de la première (le spectre est hermitien). La figure 2.25 illustre la reconstruction du spectre à partir de l'enveloppe spectrale.

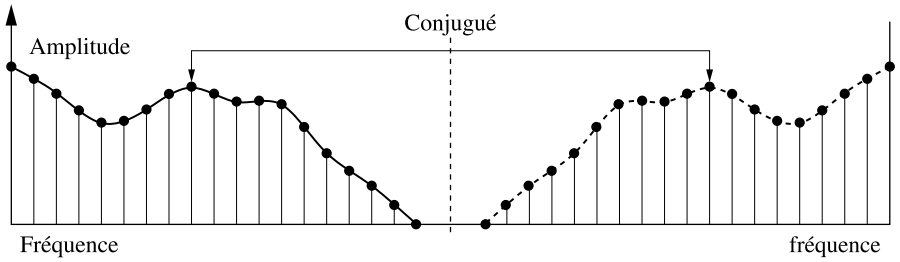


Figure 2.25. *Le spectre discret est reconstruit à partir de l'échantillonnage uniforme de l'enveloppe spectrale du bruit (pour les indices strictement positifs, puisque l'indice 0 correspond à la composante continue, toujours mise à 0). Seule la partie réelle du spectre (complexe) est représentée ici.*

Dans le cas de la voix chantée, l'utilisation de l'enveloppe spectrale de voyelles (obtenue à partir de l'analyse de voyelles voisées, voir chapitre « Méthodes d'analyse du signal musical ») pour générer le bruit permet d'obtenir de manière très réaliste les mêmes voyelles mais non voisées, comme chuchotées.

2.6.2. *Transformée de Fourier inverse*

Une fois l'approximation du spectre à court terme calculée, la transformée de Fourier rapide inverse (IFFT) peut alors être appliquée comme pour la technique FFT^{-1} de la section 2.2.

2.6.3. *Technique* overlap-add

La même technique *overlap-add* qu'en section 2.2 est alors utilisée afin d'éviter les clics. Même si la fenêtre de Bartlett (triangulaire) est souvent utilisée en pratique, nous recommandons de prendre plutôt la fenêtre de Hann, car elle réduit certains artefacts produits par la technique *overlap-add* et produit ainsi un résultat bien meilleur sur le plan perceptif.

En fait, la méthode présentée ci-dessus ne peut ajouter aux sons que des bruits blancs filtrés, comme dans le cas du modèle SMS (voir section 2.1). De plus, les parties déterministe (sinusoïdes) et stochastique (bruit) des sons produits, générées par des algorithmes différents, se retrouvent au final décorrélées, ce qui peut être très gênant. Pour certains sons bruités, il existe des méthodes de synthèse plus adaptées.

Alors que la synthèse additive consiste à superposer des oscillations quasi sinusoïdales pour obtenir un son complexe à partir du silence, la synthèse soustractive [JAF 83, KAR 83] est basée sur l'idée complémentaire qui consiste à partir d'un son complexe et d'en éliminer – par filtrage – les composantes fréquentielles indésirables. Il existe un très grand nombre de techniques de synthèse. Pour plus d'informations sur la synthèse sonore en général, nous recommandons de consulter, entre autres, les ouvrages suivants : [MOO 90, ROA 96].

2.7. Bibliographie

[ABR 66] ABRAMOWITZ M., STEGUN I.A., *Handbook of Mathematical Functions*, National Bureau of Standards, Washington, D.C., 1966.

[ALL 77a] ALLEN J.B., « Short-term spectral analysis, synthesis, and modification by the discrete Fourier transform », *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, n° 3, p. 235-238, 1977.

[ALL 77b] ALLEN J.B., RABINER L.R., « A unified approach to short-time Fourier analysis and synthesis », *Proceedings of the IEEE*, vol. 65, n° 11, p. 1558-1564, novembre 1977.

[ARF 79] ARFIB D., « Digital synthesis of complex spectra by means of multiplication of nonlinear distorted sine waves », *Journal of the Audio Engineering Society*, vol. 27, n° 10, p. 757-768, 1979.

[ASS 85] ASSAYAG G., CASTELLENGO M., MALHERBE C., Nouvelles techniques instrumentales, Rapport technique 38, IRCAM, 1985.

[BRA 95] BRANDENBURG K., BOSI M., « Overview of MPEG-audio: Current and future standards for low bit-rate audio coding », dans *Ninety-ninth Convention of the Audio Engineering Society*, prétirage 4130, Audio Engineering Society (AES), New York, octobre 1995.

[BRI 95] BRISTOW-JOHNSON R., « A detailed analysis of a time-domain formant-corrected pitch-shifting algorithm », *Journal of the Audio Engineering Society*, vol. 43, n° 5, p. 340-352, 1995.

[BEL 99] BÉLANGER D., « Data structures and algorithms: Skip lists », http://www.cs.mcgill.ca/~dbelan2/cs251/skip_lists.html, Computer Science Course, McGill University, Canada, 1999.

[CAB 76] CABOT R.C., MINO M.H., DORANS D.A., TACKEL I.S., BREED H.E., « Detection of phase shifts in harmonically related tones », *Journal of the Audio Engineering Society*, vol. 24, n° 7, p. 568-571, septembre 1976.

[CAS 94] CASTELLENGO M., « Les sources acoustiques », dans *Le livre des techniques du son*, vol. 1, p. 45-70, Editions Fréquences – Diffusion Eyrolles, Paris, seconde édition, 1994.

[CHO 73] CHOWNING J.M., « The synthesis of complex audio spectra by means of frequency modulation », *Journal of the Audio Engineering Society*, vol. 21, n° 7, p. 526-534, 1973.

[COO 65] COOLEY J.W., TUKEY J.W., « An algorithm for the Machine computation of complex Fourier series », *Mathematics of Computation*, vol. 19, p. 297-301, avril 1965.

[DOL 86] DOLSON M.B., « The phase vocoder: A tutorial », *Computer Music Journal*, vol. 10, n° 4, p. 14-27, 1986.

[EVA 98] EVANGELISTA G., CAVALIERE S., « Dispersive and pitch-synchronous processing of sounds », dans *Digital Audio Effects (DAFx) Conference*, Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research), Barcelone, Espagne, p. 232-236, novembre 1998.

[FED 98] DI FEDERICO R., « Waveform preserving time stretching and pitch shifting for sinusoidal models of sound », dans *Digital Audio Effects (DAFx) Conference*, Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research), Barcelone, Espagne, p. 44-48, novembre 1998.

[FIT 96] FITZ K., HAKEN L., « Sinusoidal modeling and manipulation using lemur », *Computer Music Journal*, vol. 20, n° 4, p. 44-59, 1996.

[FOL 95] FOLEY J.D., VAN DAM A., FEINER S.K., HUGHES J.F., « Representing curves and surfaces », dans *Computer Graphics–Principles and Practice*, chapitre 11, p. 471-531, Addison-Wesley, Reading, Massachusetts, System Programming Series, seconde édition, 1995.

[FRE 92] FREED A., RODET X., DEPALLE P., « Synthesis and control of hundreds of sinusoidal partials on a desktop computer without custom hardware », dans *Conférence ICSPAT'92*, 1992.

[FRE 93] FREED A., RODET X., DEPALLE P., « Performance, synthesis and control of additive synthesis on a desktop computer using FFT^{-1} », dans *International Computer Music Conference (ICMC)*, International Computer Music Association (ICMA), Tokyo, Japon, 1993.

[FRI 98] FRIGO M., JOHNSON S.G., FFTW user's manual, MIT, http://www.fftw.org, 1998.

[GAR 99] GARCIA G., PAMPIN J., « Data compression of sinusoidal modeling parameters based on psychoacoustic masking », dans *International Computer Music Conference (ICMC)*, International Computer Music Association (ICMA), Pékin, Chine, p. 40-43, octobre 1999.

[GOR 85] GORDON J.W., SMITH J.O., « A sine generation algorithm for VLSI applications », dans *International Computer Music Conference (ICMC)*, Vancouver, Canada, p. 165-168, 1985.

[HAR 78] HARRIS F.J., « On the use of windows for harmonic analysis with the discrete Fourier transform », dans *Proceedings of the IEEE*, vol. 66, p. 51-83, 1978.

[HEL 54] VON HELMHOLTZ H., *On the sensations of tone as a physiological basis for the theory of music*, Dover Publications, New York, 1954 (traduction de l'édition de 1877 par A.J. Ellis).

[HER 95] HERRE J., BRANDENBURG K., EBERLEIN E., GRILL B., « Second generation ISO/MPEG-audio layer III coding », dans *Ninety-eighth Convention of the Audio Engineering Society*, prétirage 3939, Paris, février 1995.

[HOD 99] HODES T., FREED A., « Second-order recursive oscillators for musical additive synthesis applications on SIMD and VLIW processors », dans *International Computer Music Conference (ICMC)*, International Computer Music Association (ICMA), Pékin, Chine, p. 74-77, octobre 1999.

[JAF 83] JAFFE D., SMITH J., « Extensions of the Karplus-Strong plucked-string algorithm », *Computer Music Journal*, vol. 7, n° 2, p. 56-69, 1983.

[JAY 93] JAYANT N., JOHNSTON J., SAFRANEK R., « Signal compression based on models of human perception », dans *Proceedings of the IEEE*, vol. 81, p. 1385-1421, octobre 1993.

[KAR 83] KARPLUS R., STRONG A., « Digital synthesis of plucked string and drum timbres », *Computer Music Journal*, vol. 7, n° 2, p. 43-55, 1983.

[KIT 94] KITANTOU M., « La perception auditive », dans *Le livre des techniques du son*, vol. 1, p. 155-181, Editions Fréquences – Diffusion Eyrolles, Paris, seconde édition, 1994.

[LAG 01a] LAGRANGE M., Accélération de la synthèse sonore, Mémoire de maîtrise, Université de Bordeaux 1, LABRI, juin 2001.

[LAG 01b] LAGRANGE M., MARCHAND S., « Real-time additive synthesis of sounds by taking advantage of psychoacoustics », dans *Digital Audio Effects (DAFx) Conference*, University of Limerick and COST (European Cooperation in the Field of Scientific and Technical Research), Limerick, p. 5-9, décembre 2001.

[MAR 99a] MARCHAND S., « Musical sound effects in the SAS model », dans *Digital Audio Effects (DAFx) Conference*, Norwegian University of Science and Technology (NTNU) and COST (European Cooperation in the Field of Scientific and Technical Research), Trondheim, p. 139-142, décembre 1999.

[MAR 99b] MARCHAND S., STRANDH R., « InSpect and ReSpect: Spectral modeling, analysis and real-time synthesis software tools for researchers and composers », dans *International Computer Music Conference (ICMC)*, International Computer Music Association (ICMA), Pékin, Chine, p. 341-344, octobre 1999.

[MAR 00a] MARCHAND S., « InSpect software package », http://www.scrime.u-bordeaux.fr/InSpect, 2000.

[MAR 00b] MARCHAND S., Sound Models for Computer Music (Analysis, Transformation, Synthesis), Thèse de doctorat, Université de Bordeaux 1, LABRI, décembre 2000.

[MAR 01a] MARCHAND S., « InSpect+ProSpect+ReSpect software packages », http://www.scrime.u-bordeaux.fr/ProSpect, 2001.

[MAR 01b] MARCHAND S., « Musical audio effects in the SAS model », *Journal of New Music Research*, vol. 30, n° 3, p. 259-269, septembre 2001.

[MAS 95] MASRI P., BATEMAN A., « Identification of nonstationary audio signals using the FFT, with application to analysis-based synthesis of sound », dans *Proceedings IEE Colloquium on Audio Engineering*, The Institution of Electrical Engineers (IEE), Londres, Grande-Bretagne, p. 11.1-11.6, 1995.

[MAS 98] MASRI P., CANAGARAJAH N., « Extracting more detail from the spectrum with phase distortion analysis », dans *Digital Audio Effects (DAFx) Conference*, Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research), Barcelone, Espagne, p. 119-122, novembre 1998.

[MAT 80a] MATHEWS M.V., PIERCE J.R., Harmony and nonharmonic partials, Rapport technique 28, IRCAM, 1980.

[MAT 80b] MATHEWS M.V., PIERCE J.R., « Harmony and nonharmonic partials », *Journal of the Acoustical Society of America*, vol. 68, p. 1252-1257, 1980.

[MCA 86] MCAULAY R.J., QUATIERI T.F., « Speech analysis/synthesis based on a sinusoidal representation », *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, n° 4, p. 744-754, 1986.

[MIDI 96] MIDI Manufacturers Association, « The complete MIDI 1.0 detailed specification–Standard MIDI files 1.1 », http://www.midi.org, 1996.

[MOO 78a] MOORE F.R., « An introduction to the mathematics of digital signal processing », *Computer Music Journal*, vol. 2, n° 1, p. 38-47, 1978 (partie 1).

[MOO 78b] MOORE F.R., « An introduction to the mathematics of digital signal processing », *Computer Music Journal*, vol. 2, n° 2, p. 48-60, 1978 (partie 2).

[MOO 85] MOORE F.R., « An introduction to the mathematics of digital signal processing », dans *Digital Audio Signal Processing*, A-R Editions, Inc, Madison, Wisconsin, The Computer Music and Digital Audio Series, p. 1-67, 1985.

[MOO 90] MOORE F.R., *Elements of Computer Music*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[MOOR 77] MOORER J.A., « Signal processing aspects of computer music–A survey », *Computer Music Journal*, vol. 1, n° 1, p. 4-37, 1977.

[MOOR 78] MOORER J.A., « The use of the phase vocoder in computer music applications », *Journal of the Audio Engineering Society*, vol. 26, n° 1/2, p. 42-45, 1978.

[MOOR 85] MOORER J.A., « Signal processing aspects of computer music: A survey », dans *Digital Audio Signal Processing*, A-R Editions, Inc, Madison, Wisconsin, The Computer Music and Digital Audio Series, p. 149-220, 1985.

[MOR 77] MORILL D., « Trumpet algorithms for computer composition », *Computer Music Journal*, vol. 1, n° 1, p. 46-52, 1977 (réédité par C. Roads et J. Strawn dans *Foundations of Computer Music*, MIT Press, Cambridge, 1985).

[NYQ 28] NYQUIST H., « Certain topics in telegraph transmission theory », *AIEE Transactions*, p. 617-644, 1928.

[OPP 89] OPPENHEIM A.V., SCHAFER R.W., *Discrete-Time Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, Signal Processing Series, 1989.

[ORF 96] ORFANIDIS S.J., *Introduction to Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, Signal Processing Series, 1996.

[PAI 01] PAINTER T., SPANIAS A., « A review of algorithms for perceptual coding of digital audio signals », http://www.eas.asu.edu~spanias, 2001.

[PAN 95] PAN D., « A tutorial on MPEG/audio compression », *IEEE Multimedia*, vol. 2, n° 2, p. 60-74, 1995.

[PAP 95] PAPAMICHALIS P., « MPEG audio compression: Algorithm and implementation », dans *Proceedings of the International Conference on DSP*, p. 72-77, juin 1995.

[PEE 98] PEETERS G., « Analyse-synthèse des sons musicaux par la méthode PSOLA », dans *Journées d'informatique musicale (JIM)*, Toulon, France, 1998.

[POR 76] PORTNOFF M.R., « Implementation of the digital phase vocoder using the fast Fourier transform », *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 24, n° 3, p. 243-248, juin 1976.

[POR 80] PORTNOFF M.R., « Time-frequency representation of digital signals and systems based on short-time Fourier transform », *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, n° 8, p. 55-69, février 1980.

[PUG 90] PUGH W., « Skip lists: A probabilistic alternative to balanced trees », *Communications of the ACM*, p. 668-676, 1990.

[RIO 84] RIOTTE A., « Un modèle informatique pour la transformation continue de sons inharmoniques », dans Buxton W. (dir.), *International Computer Music Conference (ICMC)*, Computer Music Association, San Francisco, Californie, 1984.

[RIS 86] RISSET J.-C., « Timbre et synthèse de sons », *Analyse musicale*, p. 9-19, 1986.

[RIS 88] RISSET J.-C., « Perception, environnement, musiques », *Harmoniques*, vol. 2, p. 10-42, 1988.

[RIS 91] RISSET J.-C., « Timbre analysis by synthesis: Representations, imitations, and variants for musical composition », dans *Representation of Musical Signals*, chapitre 1, p. 7-43, MIT Press, Cambridge, Massachusetts, 1991.

[ROA 96] ROADS C., *The Computer Music Tutorial*, MIT Press, Cambridge, Massachusetts, 1996.

[SCH 77] SCHOTTSTAEDT B., « The simulation of natural instrument tones using frequency modulation with a complex modulating wave », *Computer Music Journal*, vol. 1, n° 4, p. 46-50, 1977 (réédité par C. Roads et J. Strawn dans *Foundations of Computer Music*, MIT Press, Cambridge, Massachusetts, 1985).

[SCHW 99] SCHWARZ D., RODET X., « Spectral envelope estimation and representation for sound analysis-synthesis », dans *International Computer Music Conference (ICMC)*, International Computer Music Association (ICMA), Pékin, Chine, p. 351-354, octobre 1999.

[SER 97] SERRA M.-H., « Introducing the phase vocoder », dans *Musical Signal Processing*, chapitre 2, p. 31-90, Swets et Zeitlinger, Lisse, Pays-Bas, Studies on New Music Research, 1997.

[SERR 89] SERRA X., A System for Sound Analysis/Transformation/Synthesis Based on a Deterministic plus Stochastic Decomposition, Thèse de doctorat, CCRMA, Département de musique, Université de Stanford, 1989.

[SERR 90] SERRA X., SMITH J.O., « Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition », *Computer Music Journal*, vol. 14, n° 4, p. 12-24, 1990.

[SERR 97] SERRA X., « Musical sound modeling with sinusoids plus noise », dans *Musical Signal Processing*, chapitre 3, p. 91-122, Swets et Zeitlinger, Lisse, Pays-Bas, Studies on New Music Research, 1997.

[SHA 49] SHANNON C.E., « Communication in the presence of noise », dans *Proceedings of IRE*, p. 10-12, janvier 1949.

[SMI 84] SMITH J.O., GOSSETT P., « A flexible sampling-rate conversion method », dans *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, San Diego, Californie, vol. 2, p. 19.4.1-19.4.2, mars 1984.

[SMI 92] SMITH J.O., COOK P.R., « The second-order digital waveguide oscillator », dans *International Computer Music Conference (ICMC)*, San Jose, Californie, p. 150-153, 1992.

[SMI 00] SMITH J.O., Digital audio resampling home page, Rapport technique, CCRMA, http://www-ccrma.stanford.edu/~jos/resample/resample.html, 2000.

[STR 99] STRANDH R., MARCHAND S., « Real-time generation of sound from parameters of additive synthesis », dans *Journées d'informatique musicale (JIM)*, CEMAMU, Paris, France, p. 83-88, mai 1999.

[VER 98a] VERMA T.S., MENG T.H.Y., « An analysis/synthesis tool for transient signals that allows a flexible sines+transients+noise model for audio », dans *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, mai 1998.

[VER 98b] VERMA T.S., MENG T.H.Y., « Time scale modification using a sines+transients+noise signal model », dans *Digital Audio Effects (DAFx) Conference*, Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research), Barcelone, Espagne, p. 49-52, novembre 1998.

[WEG 24] WEGEL R.L., LANE C.E., « The auditory masking of one pure tone by another and its probable relation to the dynamics of the inner ear », *Physical Review*, vol. 23, p. 266-285, 1924.

[ZWI 81] ZWICKER E., FELDTKELLER R., *Psychoacoustique – L'oreille, récepteur d'information*, Masson, Paris, 1981.

[ZWI 90] ZWICKER E., FASTL H., *Psychoacoustics Facts and Models*, Springer-Verlag, New York, 1990.

[ZWI 91] ZWICKER E., ZWICKER U.T., « Audio engineering and psychoacoustics: Matching signals to the final receiver, the human auditory system », *Journal of the Audio Engineering Society*, vol. 39, n° 3, p. 115-126, mars 1991.

Chapitre 3

Techniques de spatialisation des sons

3.1. Introduction

Hauteur, durée et intensité sont les attributs traditionnels de la notation musicale. Lorsque le compositeur choisit ses instruments en spécifiant une orchestration, le timbre en est un quatrième. La mise en espace des sons, ou spatialisation, est un attribut moins (re)connu et pourtant utilisé et annoté depuis plus de quatre siècles.

Pour comprendre ce que recouvre la spatialisation des sons, interrogeons-nous sur ce qui habite la notion d'espace sonore dans notre expérience quotidienne. Un espace peut être un lieu clos, comme une salle, ou un espace ouvert, comme une ville, une forêt, un bord de mer. Lorsque nous traversons ces espaces, les sons nous parviennent de multiples incidences. Celles que nous distinguons le mieux sont les sources sonores qui émettent en notre direction et qui produisent ce que nous appellerons le « son direct ». Mais nous percevons également les multiples réflexions de ce son direct sur les obstacles qu'il rencontre lors de son trajet de son point d'émission jusqu'à nos tympans. Ces réflexions constituent un champ sonore réfléchi et réverbéré, que l'on englobe souvent sous le vocable « effet de salle » bien qu'il soit également caractéristique d'espaces ouverts. Ces deux notions, son direct et effet de salle, sont indissociables pour décrire un espace sonore ou le simuler, c'est-à-dire spatialiser les sons. Aussi, par spatialisation entendons-nous le choix de la position des sources sonores dans l'espace et celui de l'acoustique du lieu dans lequel ces sons sont projetés.

Chapitre rédigé par Véronique LARCHER ET Arnaud LABORIE.

Dans ce chapitre, nous illustrons l'intérêt des compositeurs pour la spatialisation des sons et présentons les techniques les plus utilisées pour donner l'illusion de sources sonores provenant de toute incidence. Les approches permettant de modéliser et reproduire l'effet de salle sont décrites par exemple dans [JOT 92] et [JUL 94].

3.2. Rôle de la spatialisation dans la composition musicale

Dans cette section, nous décrivons des explorations musicales marquantes de la spatialisation. Certaines sont l'œuvre de compositeurs qui, « livrés à eux-mêmes », utilisent les contraintes naturelles de la géométrie et de l'acoustique d'un lieu. D'autres illustrent la démultiplication des possibilités de synthèse et de contrôle de la spatialisation offertes par les technologies de l'enregistrement, de l'informatique et de la diffusion électro-acoustique.

3.2.1. *Mise en espace « acoustique »*

Pour dater l'utilisation la plus ancienne de la spatialisation musicale, on mentionne souvent Andrea Gabrieli, et plus généralement l'école vénitienne du XVI^{e} siècle. Dans ses œuvres pour chœurs écrites pour la basilique Saint Marc, Gabrieli porte une attention particulière à la répartition des masses sonores dans l'espace. Il amorce un décloisonnement de la performance des musiciens en écrivant ses œuvres pour deux ensembles « orgue et chœur » qui se répondent, comme en écho l'un de l'autre, de part et d'autre de la nef. Il introduit ainsi une sensation de mouvement entre la droite et la gauche, un effet stéréo jusqu'alors inconnu pour l'auditoire.

En 1729, avec *La passion selon Saint Matthieu*, Bach ajoute un nouvel élément à la spatialisation. Brisant la frontalité de l'effet stéréo décrit plus haut, il écrit une partition pour trois chœurs, dont un chœur d'enfants qu'il veut situé derrière l'auditoire, en plus des chœurs d'adultes à droite et à gauche. Lorsque les voix de ces trois chœurs se mêlent, le son entoure et enveloppe les auditeurs. Toutefois, la zone frontale demeure prépondérante dans la mesure où le chœur d'enfants accompagne les orchestres placés auprès des deux chœurs d'adultes.

Dans *Gruppen* (1955-1957), Karlheinz Stockhausen place trois orchestres en fer à cheval autour des auditeurs et donne ainsi au public la perspective du chef d'orchestre au centre d'un orchestre symphonique. Dans cette pièce, contrairement à la *Passion* prédemment évoquée, chacune des trois zones spatiales a le même « poids » : trois formations instrumentales de grande taille qui requièrent chacune un chef d'orchestre. Stockhausen parvient avec *Gruppen* à donner l'illusion de sons en

« mouvement », sonorités qui se transmettent d'un orchestre à l'autre dans une grande continuité.

Enfin, si la spatialisation a été utilisée pour rompre avec le mode d'écoute frontal en éclatant les sources musicales en plusieurs positions autour de l'auditoire, certaines tentatives ont poussé son utilisation jusqu'à disséminer des instruments au sein des auditeurs. C'est le cas de *Quodlibet* (1989-1991), œuvre d'Emmanuel Nunes écrite pour le Coliseu de Lisbonne. Dans cette pièce, l'effectif instrumental est divisé entre un orchestre, en fosse, sept instrumentistes assignés à la scène, et un ensemble de vingt et un instrumentistes mobiles, dispersés dans les balcons. La mise en espace a ici pour objectif de créer un théâtre purement musical, « sans geste ni parole ». Elle est par exemple utilisée pour contredire l'association couramment faite entre une position haute et un son aïgu, une position basse et un son grave.

Ces exemples musicaux, loin de présenter une liste exhaustive, illustrent différentes utilisations de la spatialisation sonore comme élément structurel de la partition depuis le XVI^{e} siècle. Tant que cette spatialisation s'appuie sur des sources sonores réelles, les compositeurs doivent se soumettre à l'acoustique des lieux dans lesquels ils jouent, et les sons ne peuvent être placés qu'en des positions physiquement atteignables. L'électro-acoustique et l'informatique ont démultiplié ces possibilités en permettant de créer des positions « virtuelles » et des ambiances sonores de synthèse.

3.2.2. *Apport de l'électro-acoustique et de l'informatique sur la spatialisation*

Avec le développement des transducteurs (microphones, haut-parleurs) et des supports d'enregistrements, il est devenu possible de suggérer des sources sonores en des positions ne coïncidant pas avec celle de la source réelle du son.

Le moyen le plus simple consiste à placer un haut-parleur en chacune de ces positions. L'une des premières œuvres utilisant une diffusion électro-acoustique multicanale a été réalisée pour le cinéma en 1939. Il s'agit de Fantasia, de Walt Disney, pour laquelle l'orchestre a été enregistré par trente-trois microphones avec pour intention de reproduire le tout sur quatre-vingt dix haut-parleurs placés derrière l'écran et entourant les spectateurs.

La spatialisation électro-acoustique est un terrain qui pendant les années 1950 a également suscité beaucoup de curiosité dans le domaine musical. L'un des exemples marquant reste *Poème électronique* , œuvre écrite par Edgard Varèse en 1958. Cette pièce a été conçue pour le pavillon Philips à l'occasion de l'exposition universelle de Bruxelles. Elle mettait en jeu plus de quatre cent haut-parleurs et a été entendue par plus de deux millions d'auditeurs.

Avec la musique acousmatique, l'utilisation de haut-parleurs, souvent associés par paires, comme « projecteurs de sons » indépendants, avec des couleurs propres à chacun, devient une technique de spatialisation codifiée [DHO 88]. Ces compositeurs, tels que François Bayle ou Francis Dhomont, composent une bande sonore stéréo en studio à partir de sons enregistrés, éventuellement transformés, ou de sons synthétiques, qu'ils spatialisent en concert par la diffusion sur un orchestre de haut-parleurs tels que l'acousmonium, inauguré en 1974 à Paris. La spatialisation est ici utilisée pour créer « l'espace externe » du son, en révélant la structure de l'œuvre et les profondeurs de champ inscrites sur le support stéréo grâce à la démultiplication appropriée des deux canaux initiaux.

Bien entendu, si l'on souhaite simuler de nombreuses positions ou bien simuler un mouvement continu de sources sonores, cette première solution consistant à placer un haut-parleur en chaque position cible devient lourde à mettre en œuvre, tant d'un point de vue économique que pratique. Le défi de la spatialisation est donc aujourd'hui de reproduire une image spatiale à partir d'un nombre limité de canaux d'enregistrement et de diffusion.

Dans *Kontakte* (1958-1960), Stockhausen a recours à une astuce pour enregistrer une source sonore décrivant une trajectoire circulaire et la reproduire sur quatre haut-parleurs (seulement) entourant l'auditoire. Il prend pour source sonore un haut-parleur qu'il rend très directif à l'aide d'un cône fixé autour du dôme du haut-parleur, puis il place l'ensemble sur une table tournante. Il entoure cette table de quatre microphones, immobiles, et enregistre le son émis par le haut-parleur alors qu'il est entraîné par la table en rotation. La piste enregistrée par chacun des microphones est destinée à être diffusée par un haut-parleur positionné dans le lieu d'écoute au même azimut que le microphone concerné. Cette expérience est spécifique au mouvement circulaire et au dispositif de reproduction utilisé (quatre haut-parleurs).

Le compositeur John Chowning a joué un rôle déterminant pour définir des moyens de créer par synthèse l'illusion de sources sonores décrivant des mouvements génériques, avec une recherche poussée de réalisme. C'est une démarche qu'il applique en 1977 dans son œuvre *Turenas* (anagramme de Natures), dans laquelle il parvient à reproduire l'effet *Doppler* pour accompagner chaque source en mouvement. L'effet *Doppler* est effectivement l'une des caractéristiques fondamentales des sources naturelles en mouvement, par lequel la fréquence des sons émis varie en fonction de la vitesse de déplacement de la source [CHO 71]. Plusieurs techniques permettant de simuler la position et le mouvement de sources sonores avec un nombre limité de haut-parleurs ou bien sur casque d'écoute sont décrites dans les sections suivantes.

3.3. Perception de l'incidence de sources sonores dans l'espace

Dans cette section, et dans celles qui suivent, nous nous concentrons sur le positionnement angulaire de sources sonores en « champ libre », c'est-à-dire sans effet de salle. Leur position angulaire est décrite par un angle d'azimut dans le plan horizontal, et par un angle repérant leur altitude, ou angle d'élévation.

Les indices de localisation constituent les clés acoustiques permettant à notre système auditif d'identifier l'incidence d'une source sonore. Ils sont de deux natures (indices interauraux/indices monauraux) et possèdent dans les deux cas des attributs temporels et fréquentiels variant avec l'incidence. La connaissance de ces indices laisse entrevoir la possibilité de « duper l'oreille » en reproduisant artificiellement une sensation de localisation.

3.3.1. *Les indices acoustiques de localisation*

Pour localiser une source, notre système auditif s'appuie en premier lieu sur les différences sonores entre les deux oreilles, que l'on désigne par « indices interauraux ». La théorie duplex de la localisation [RAY 07], met en évidence l'importance des différences interaurales de temps (ITD) et d'intensité (ILD). Toutefois, ITD et ILD ne suffisent par pour caractériser une position en azimut et en élévation, puisqu'en effet des incidences différentes peuvent produire des indices interauraux identiques. Le lieu de ces incidences est qualifié de « cône de confusion ». Pour résoudre cette ambiguité, le système auditif a recours à des indices consignés indépendamment dans les signaux parvenant à chaque oreille. Il s'agit des « indices monauraux ».

3.3.1.1. *Les indices interauraux*

L'ITD représente le décalage temporel observé entre les signaux parvenant aux oreilles. D'après les données expérimentales de Blauert, il varie entre 0 s pour les sons placés à égale distance des deux oreilles et 800 µs pour les sons placés à l'extrême droite ou gauche de l'auditeur, soit à un azimut de ± 90° [BLA 97]. L'ITD obéit à deux mécanismes du système auditif, en jeu sur deux intervalles fréquentiels distincts.

En basses fréquences (au-dessous de 1 500 Hz), l'ITD représente le retard de phase entre les signaux droite et gauche [KUH 77]. Les données expérimentales de Kuhn montrent que dans cette région fréquentielle, l'ITD est indépendant de la fréquence et peut être approché par :

$$\mathrm{ITD} \approx 3\frac{a}{c}\sin(az)$$

où a désigne le rayon d'une tête sphérique approchant elle-même une tête humaine (≈ 9,3 cm dans [KUH 77]), c la célérité du son dans l'air (340 m/s), et az l'azimut.

Au-dessus de 1 500 Hz, les retards de phase deviennent trop faibles pour expliquer l'intervalle de valeurs couvert par ITD. Celles-ci sont alors interprétées comme le décalage temporel entre l'enveloppe des signaux droite et gauche, que le système auditif estimerait par intercorrélation de ces deux signaux. Pour les fréquences supérieures à 3 kHz, Kuhn montre que l'ITD hautes-fréquences peut être estimé par une valeur indépendante de la fréquence donnée par :

$$\mathrm{ITD} \approx 2\frac{a}{c}\sin(az)$$

L'ILD quant à lui peut être exprimé pour chaque fréquence comme la différence des spectres d'énergie droite et gauche (en décibels), que l'on trouve souvent approximée par [JOT 95] :

$$\mathrm{ITD} \approx 10.\log 10\left(\frac{\int mag_L^2(f)\,.\,df}{\int mag_R^2(f)\,.\,df}\right)$$

où mag_L et mag_R désignent le spectre d'amplitude du signal reçu respectivement à l'oreille gauche ou à l'oreille droite.

Dans son modèle de l'ILD, Gaik propose une évaluation de l'ILD pour chaque bande critique, l'intégration les spectres d'énergie portant ainsi sur la largeur de chaque bande [GAIK 93]. Les données expérimentales montrent que l'ILD peut atteindre 30 dB pour les positions les plus latérales [BLA 97].

ITD et ILD caractérisent le degré de latéralisation des sources sonores, c'est-à-dire leur écartement à droite ou à gauche par rapport au centre de la tête. Plusieurs expériences montrent qu'ils se complètent sans redondance, chacun prenant la prépondérance sur des intervalles de fréquences différents. Wightman et Kistler, par exemple, montrent que l'ITD basses-fréquences est l'indice de latéralisation dominant pour les sons large bande [WIG 92]. Pour simuler un son à un certain degré de latéralisation, si ce signal a un support fréquenciel suffisemment grand, il suffirait donc de créer deux canaux sonores décalés temporellement d'une valeur correspondant aux approximations décrites plus haut. Toutefois, l'ITD perd sa prépondérance en faveur de l'ILD si la localisation est réalisée sur un son contenant peu d'énergie au-dessous de 2 000 Hz [BLA 97, KUH 77]. Pour de tels sons, la latéralisation est reproduite en introduisant des différences d'intensité conformes aux abaques fournies par les données expérimentales de la littérature [BLA 97], ou bien estimées à partir de mesures comme nous le décrivons plus bas. Il est toutefois

utile de garder à l'esprit que la relation entre les indices interauraux, que Gaik qualifie de « relation naturelle », est elle-même un indice important pour la perception du timbre de la source sonore [GAIK 93]. La reproduction partielle de ces indices (ILD sans ITD ou ITD sans ILD) peut ainsi entraîner une coloration, affectant l'extériorisation du son, sa reconnaissance voire même sa plausibilité.

3.3.1.2. *Les indices monauraux*

Comme l'illustre la notion de cône de confusion, les indices interauraux sont caractéristiques de la latéralisation, mais ne permettent pas de discriminer l'avant de l'arrière. Pour résoudre cette indétermination, le système auditif a recours à des indices dits monauraux, extraits des signaux droit et gauche indépendamment. Comme pour les indices interauraux, le système auditif interprète l'information monaurale en pratiquant conjointement une analyse dans le domaine temporel et une analyse dans le domaine fréquentiel.

Le codage temporel des indices monauraux s'appuie sur l' « effet de précédence », phénomène par lequel le premier front d'onde conditionne la localisation d'une source sonore, quelque soit l'incidence des réflexions lui succédant après 80 ms ou au-delà [GAR 68]. On peut penser que le système auditif est sensible aux réflexions du son sur le pavillon, qui parviennent en deçà de cette limite temporelle et qui varient en fonction de l'incidence du son. Batteau avance ainsi que l'information monaurale de localisation est codée sur un intervalle de 350 µs après l'arrivée du son direct [BAT 67]. Sur cet intervalle, le pavillon engendrerait deux principales réflexions dont l'écart temporel avec le son direct caractériserait l'incidence de la source.

Le codage fréquentiel des indices monauraux est la conséquence des transformations subies par le son incident sur le corps de l'auditeur (oreilles, tête, torse). Ces transformations engendrent des phénomènes de résonances amplifiant sélectivement certaines bandes de fréquences. Blauert montre ainsi l'existence de « bandes directionnelles », intervalles fréquentiels qui, lorsqu'ils sont riches en énergie, induisent une localisation dans une zone d'incidence déterminée [BLA 97]. C'est ainsi par exemple qu'un son riche en énergie autour de 8 kHz sera perçu comme provenant d'une incidence élevée. Le codage de l'incidence consisterait donc pour partie à régler le rapport d'énergie entre les bandes directionnelles.

Les indices monauraux présentés plus haut sont de même nature que les indices permettant d'identifier le timbre de la source (analyse temporelle, analyse spectrale). Pourtant, le système auditif parvient à isoler les deux phénomènes : nous savons percevoir le mouvement d'une source sonore sans interpréter les variations de ses caractéristiques temporelles et fréquentielles comme une variation de timbre. En vérité, cette séparation est bien meilleure pour des sources sonores familières que pour des sources auxquelles l'auditeur n'a pas été préalablement soumis. Dans ce dernier cas, Blauert observe une dégradation des performances de localisation,

puisqu'une partie des indices monauraux sont injustement attribués aux caractéristiques timbrales du son.

3.3.1.3. *Mesure des indices de localisation*

Les indices de localisation sont entièrement contenus par la fonction de transfert entre le point d'émission du son et les tympans de l'auditeur. Pour chaque point d'émission (c'est-à-dire pour chaque incidence), ces fonctions de transfert consignent l'empreinte des transformations subies par le son lors de sa propagation, et plus précisément les effets de diffraction, de diffusion et de réflexion sur le corps de l'auditeur. Elles dépendent de l'incidence du son et des caractéristiques morphologiques de l'individu, et sont couramment désignées par *Head-Related Transfer Functions* (HRTF).

Plusieurs jeux de HRTF sont librement accessibles, par exemple, celles réalisées par Algazi *et al.* à UC Davis sur de nombreuses têtes humaines pour plus de mille incidences chacune [AAT 99], ou celles de Bill Gardner et Keith Martin réalisées au MIT Medialab sur la tête artificielle KEMAR pour 710 incidences [GM 94].

Les HRTF mesurées contiennent beaucoup d'information. Pour concentrer cette dernière aux indices de localisation que nous souhaitons caractériser et reproduire, un certain nombre de post-traitements doivent être appliqués, comme l'égalisation des transducteurs de mesure et du conduit auditif [LAR 98, MOL 92].

La figure 3.1 présente un exemple de valeurs prises par les indices de localisation. Les indices monaurauraux sont constitués des HRTF mesurées aux oreilles droites et gauches à 45°, après post-traitement.

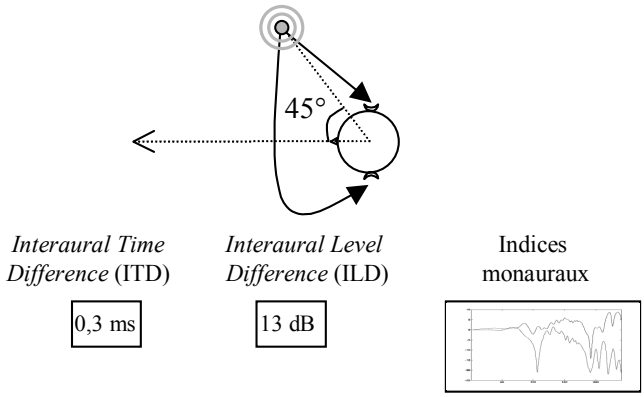


Figure 3.1. *Exemple de valeurs prises par les localisation (azimut = 45°, élévation = 0°)*

3.3.2. *Indices de localisation et techniques de spatialisation*

Pour positionner un son dans l'espace, les techniques de spatialisation commencent par transformer la source en un certain nombre de canaux sonores consignant des indices spatiaux caractéristiques de la position cible. Cette transformation est l'étape d'encodage. Les techniques décrites ci-après se distinguent sur les indices spatiaux qu'elles encodent. Certaines techniques encodent les indices de localisation entiers, d'autres les « simplifient » ou ne s'appuient que sur certains d'entre eux.

A l'étape d'encodage correspond une étape de décodage, traitement permettant de restituer l'empreinte spatiale encodée sur un dispositif de diffusion spécifique (casque ou haut-parleurs).

3.4. Les techniques binaurales et transaurales

Il faut remonter aux années 1920 pour trouver les premiers tests de spatialisation effectués avec les techniques binaurales aux Bell Laboratories (New Jersey, Etats-Units). Ces techniques ont pour objectif de reproduire aux tympans de l'auditeur le champ acoustique présent en situation d'écoute réelle. Si les causes naturelles de la localisation sont ainsi recréées, le système auditif peut s'appuyer sur les indices de localisation qui lui sont familiers et recouvrer l'incidence des sources sonores. Le système de diffusion utilisé par les techniques binaurales est le casque d'écoute. Les techniques transaurales quant à elles s'appuient sur le même encodage, mais adaptent le contenu binaural à une restitution sur une paire de haut-parleurs.

3.4.1. *Encodage des indices de localisation*

L'encodage de la position des sources sonores peut être réalisé par enregistrement ou par synthèse. Dans les deux cas, il est possible de choisir entre l'encodage d'indices « individuels », c'est-à-dire caractéristiques d'un individu particulier, et l'encodage d'indices « typiques » représentant une moyenne d'individus, le plus souvent obtenus avec une tête artificielle. Dans le premier cas, des capteurs sont insérés dans les conduits auditifs de l'individu considéré. Comme le discute en détails Moller [MOL 92], il peut s'agir de sondes glissées jusqu'au niveau du tympan, ou plus couramment des microphones à électret fixés à l'entrée du conduit auditif.

3.4.1.1. *Les enregistrements binauraux*

Traditionnellement, les enregistrements binauraux consistent à capturer les signaux en sortie des microphones insérés dans les conduits auditifs. Les deux canaux enregistrés consignent alors les indices de localisation spécifiques à la tête considérée.

Des méthodes alternatives ont été développées afin d'encoder des indices directionnels « neutres » et de n'ajouter les indices fréquentiels, et avec eux les particularités individuelles, qu'à l'étape de décodage. Avec ces approches, par exemple le Binaural B présenté dans [JOT 98], la position des sources sonores est encodée à l'aide de différences de temps et d'intensité indépendantes de la fréquence. Le Binaural B s'appuie sur un couple de microphones non coïncidents, chacun de ces microphones étant constitué d'une capsule omnidirectionnelle et de trois capsules birectionnelles. Ces approches alternatives permettent ainsi d'obtenir des enregistrements « universels », mais requièrent l'enregistrement de nombreux canaux (huit dans le cas du binaural B).

Qu'il s'agisse d'enregistrements sur tête artificielle, sur tête humaine ou bien d'enregistrements alternatifs de type Binaural B utilisant au moins deux capteurs disjoints, la distribution des sources sonores est fixée et compromet toute manipulation *a posteriori* de leur position relative.

3.4.1.2. *La synthèse binaurale*

Si l'on souhaite offrir à l'auditeur la possibilité d'interagir avec la scène sonore (se déplacer dans la scène sonore, par exemple), ou bien si cette scène sonore ne peut être capturée *live*, on a alors recours à la synthèse binaurale.

Une scène sonore peut être créée à partir de signaux élémentaires monophoniques. Comme l'illustre la figure 3.2, leur position est reproduite en filtrant chacun des signaux monophoniques par les filtres binauraux droite et gauche (HRTF) mesurés à cette position. Ce principe, simple, s'accompagne de plusieurs arbitrages à réaliser pour la mise en pratique de la synthèse binaurale.

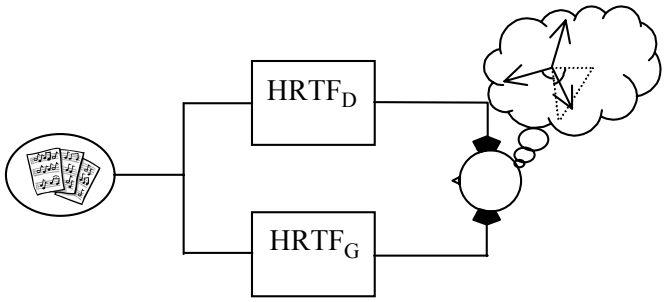


Figure 3.2. *Schéma de principe de la synthèse binaurale*

Le premier porte sur le choix des positions à synthétiser. Il est parfois difficile d'obtenir des mesures pour chacune d'entre elles, notamment si l'on a recourt à une base de données de filtre pré-existente. La solution de facilité consiste à se rabattre

sur la position mesurée la plus proche. Si l'écart est perceptible, ou bien si l'on souhaite faire évoluer la position de la source de façon continue entre deux positions mesurées, on a alors recourt à des techniques d'interpolation des HRTF [CHE 96, HAR 99, JOT 95, LAR 97].

Une autre considération pratique concerne la modélisation des HRTF. Un modèle couramment utilisé pour l'implantation de la synthèse binaurale consiste à décomposer les HRTF en un retard pur, qui représente le retard interaural, et en un filtre à phase minimale [MEH 77, WIG 92]. Le coût de calcul consenti pour l'implantation de cette composante à phase minimale varie en fonction du nombre de sources sonores que l'on souhaite spatialiser et a un impact direct sur la qualité du résultat.

3.4.2. *Décodage pour une écoute au casque ou sur haut-parleurs*

La restitution des signaux binauraux sur un casque d'écoute requiert une correction spectrale pour compenser la contribution fréquentielle des écouteurs. L'égalisation du casque s'appuie sur la mesure de sa fonction de transfert avec la même chaîne de captation (oreilles, microphone) que celle utilisée lors de l'encodage binaural.

La diffusion sur haut-parleurs introduit deux modifications majeures, tenant au fait que, contrairement au cas du casque, les transducteurs sont ici distants de l'oreille. La première conséquence est qu'à l'encodage spatial initial des signaux binauraux se superpose une seconde empreinte spatiale, caractéristique de la position des haut-parleurs (les signaux binauraux passent une seconde fois « au travers l'oreille »). De plus, le signal binaural gauche, initialement destiné à l'oreille gauche seule, parvient également à l'oreille droite et se superpose au signal binaural droite et par là même brouille les indices directionnels encodés (et réciproquement).

Le décodeur transaural est destiné à compenser ces deux effets. Il nécessite la mesure des deux fonctions de transfert binaurales correspondant aux positions des haut-parleurs et de l'auditeur dans le lieu d'écoute. Les quatre fonctions de transfert ainsi obtenues forment la matrice de transfert qui caractérise la transformation des signaux Z_g et Z_d émis par les haut-parleurs vers les signaux Y_g et Y_d reçus par les conduits auditifs [JOT 95] :

$$\begin{bmatrix} Y_g \\ Y_d \end{bmatrix} = \begin{bmatrix} H_{gg} & H_{dg} \\ H_{gd} & H_{dd} \end{bmatrix} . \begin{bmatrix} Z_g \\ Z_d \end{bmatrix}$$

Le décodage transaural réalise la matrice de transfert inverse de la matrice précédente, ce qui conduit naturellement à une implémentation sous la « forme treillis » proposée par Cooper et Bauck [COO 89] :

$$\begin{bmatrix} Z_g \\ Z_d \end{bmatrix} = \frac{\begin{bmatrix} H_{dd} & -H_{dg} \\ -H_{gd} & H_{dd} \end{bmatrix}}{H_{gg} \cdot H_{dd} - H_{dg} \cdot H_{gd}} \begin{bmatrix} Z_g \\ Z_d \end{bmatrix}$$

3.4.3. *Principales applications musicales*

La qualité de spatialisation offerte par les techniques binaurales les rend privilégiées pour l'évaluation ou la simulation de la qualité acoustique des salles de concert. Des têtes artificielles sont ainsi souvent utilisées pour caractériser les salles à l'aide de réponses impulsionnelles binaurales, mesurées dans le lieu puis chargées dans un moteur de convolution temps réel.

La spécificité de ces techniques pour une écoute au casque étend également le domaine d'applications de ces techniques. En effet, le casque limite le couplage avec l'environnement extérieur. Il écarte ainsi les transformations induites par l'effet de salle du lieu d'écoute, et supprime tout risque de boucle électro-acoustique. La synthèse binaurale est donc une technique de spatialisation particulièrement appropriée pour les applications où l'utilisateur est amené à parler, par exemple pour la visioconférence, où la spatialisation renforce la discrimination des locuteurs distants, et donc l'intelligibilité de leur message.

L'utilisation d'un casque d'écoute favorise également le sentiment d'immersion pour les applications ne simulant qu'un champ visuel limité, et rend à l'audition sa spécificité de sens d'alerte mise a profit pour les applications telles que les jeux vidéos, les simulateurs de vol, de conduite, etc.

Enfin, le casque reste le dispositif d'écoute le plus discret et connaît à ce titre une popularité grandissante auprès d'individus souhaitant regarder des films ou écouter de la musique sans générer de pollution sonore. Les traitements binauraux sont appliqués au signal pour simuler des « haut-parleurs virtuels ». Le paradigme des haut-parleurs virtuels consiste à reproduire sur écouteurs des signaux encodés pour une reproduction multi-haut-parleurs, par exemple un signal au format 5.1. Chacun des canaux est considéré comme une source sonore immobile, que l'on positionne par synthèse binaurale à l'angle présumé du haut-parleur qu'il doit alimenter (par exemple ± 30°, 0° et ± 110° pour un format 5.1). Ces traitements se trouvent le plus

souvent ou bien sur des amplificateurs multicanaux, par exemple avec les technologies *Dolby Headhone* ou VMax. Ils sont également disponibles sur PC, comme avec par exemple le *plug-in* Hearo d'AKG ou les technologies CMSS disponibles sur les cartes sons de Creative Labs. Enfin, elles sont parfois embarquées avec un casque d'écoute, comme pour le DSP Pro de Sennheiser.

Plus généralement, avec le paradigme des haut-parleurs virtuels, les techniques binaurales sont utilisées pour la production d'enregistrements musicaux dont la partition électronique a initialement été conçue pour une diffusion sur plus de deux haut-parleurs. Si la diffusion de concert peut s'accommoder de multiples canaux de diffusions, le transfert sur un support usuel tel que le CD requiert de concentrer l'information spatiale sur deux canaux. C'est le résultat que permet d'obtenir la binauralisation du format multicanal de départ, tout en conservant la spatialité souhaitée par le compositeur.

3.4.4. *Performances et limites*

En dépit des ambitions affichées par les techniques binaurales, leurs performances sont influencées par plusieurs facteurs.

Le système auditif a des performances limitées, notamment en termes de résolution angulaire pour les positions latérales ou en hauteur qui, dans des situations d'écoute réelle, sont compensés par des indices visuels ou cognitifs. Blauert rappelle par exemple qu'une source sonore d'incidence 90º dans le plan horizontal sera perçue légèrement plus en avant, vers 80º, avec un écart type de ± 9º [BLA 97]. Les techniques binaurales ne peuvent certes dépasser les limites inhérentes au système qu'elles cherchent à reproduire.

En outre, les variations individuelles des indices de localisation compromettent la possibilité d'une restitution spatiale fidèle lorsque l'auditeur n'utilise pas « ses oreilles », c'est-à-dire si les enregistrements binauraux n'ont pas été réalisés sur sa tête ou bien si les HRTF utilisées ne sont pas ses propres filtres. Ne pas individualiser les indices binauraux peut se traduire par une altération de la localisation des sources sonores (distorsion de l'azimut perçu, augmentation du nombre de confusions avant-arrière, et de sons perçus à l'intérieur de la tête) et une forte coloration du timbre de la source sonore. Comme on peut le voir sur la figure 3.3, si un individu à « petite tête » (courbes les plus à l'intérieur) utilise l'ITD d'une « grosse tête », la distorsion de l'azimut perçu peut atteindre 20°. Dans les applications grand public, pour des raisons pratiques évidentes, il est pourtant courant de ne trouver que des indices binauraux génériques.

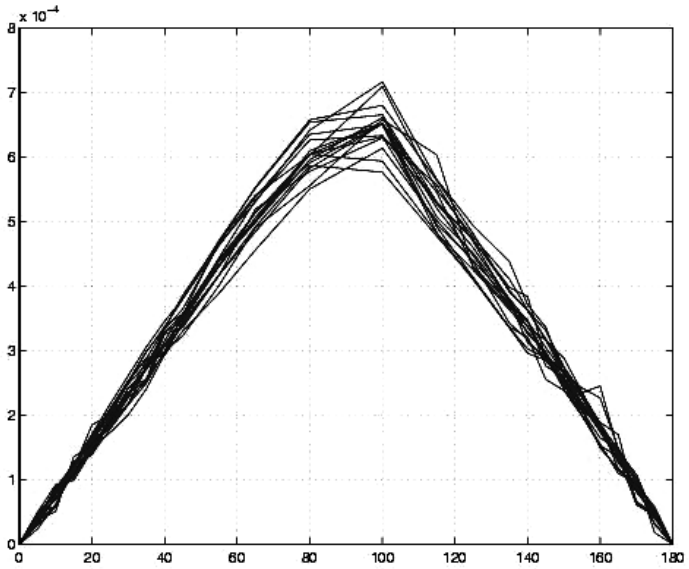


Figure 3.3. *Retard interaural (ITD) estimé à partir de HRTF mesurées sur 17 têtes*

Enfin, les sacrifices réalisés dans la modélisation des HRTF est un autre facteur limitant. A ce titre, on peut considérer que les performances de la synthèse binaurale sont le plus souvent inférieures à celles obtenues avec des enregistrements binauraux. Les tests décrits dans la littérature montrent par exemple que pour un système de synthèse binaurale non individuel, s'appuyant sur un modèle de HRTF constitué d'un retard pur et d'un filtre IIR à phase minimale d'ordre 20,33 % des sons sont perçus à l'intérieur de la tête et qu'un quart des sons perçus à l'extérieur de la tête sont l'objet de confusions avant-arrière. L'azimut est perçu avec un biais, minimum pour les sources autour de 0°, 180° et autour de 90°, mais qui atteint 20° pour les positions intermédiaires, surtout à l'avant. Ce phénomène s'accompagne d'une forte erreur sur la perception de l'élévation des sources.

Plusieurs approches ont été développées pour améliorer ces performances. Il est tout d'abord important de bien choisir la tête utilisée dans le cas d'une synthèse binaurale non individuelle. Certains préconisent l'utilisation d'une tête artificielle aux caractéristiques morphologies moyennes [BUR 75, BURA 91], tandis que d'autres recommandent l'utilisation des HRTF d'un « bon localisateur humain » [WEN 91]. Certains produits offrent aussi la possibilité de choisir partir plusieurs têtes (ou jeux de HRTF) comme le DSP Pro de Sennheiser. D'autres technologies permettent même à l'utilisateur d'ajuster les modèles de HRTF à leur morphologie, par exemple la technologie *Virtual Ear* de Sensaura.

Pour la synthèse binaurale toujours, l'ajout d'un système de suivi de position de la tête, mettant à jour la position des HRTF afin de simuler une scène sonore immobile, indépendante des mouvements de l'auditeur, améliore de façon significative les performances et réduit notamment les occurrences de confusions avant-arrière.

L'ajout d'un effet de salle, enfin, contribue à améliorer l'extériorisation des signaux binauraux et donc à réduire l'occurrence de perception des sons à l'intérieur de la tête. C'est l'approche utilisée par exemple par le système de virtualisation *Dolby Headphone*, qui altère le contenu sonore initial en ajoutant une réverbération artificielle, mais virtualise ainsi avec plus de succès les contenus 5.1.

Les techniques transaurales souffrent des mêmes limites liées à la modélisation et à l'absence d'individualisation des HRTF. De nombreux systèmes grand public parviennent toutefois à reproduire de façon convaincante des sources sonores sur les côtés, jusqu'à environ ± 90°, les positions plus à l'arrière se repliant souvent vers l'avant. C'est le cas par exemple du Spatialisateur de l'Ircam [JOT 95] ou des cartes son pour PC Audigy de Creative Labs [JOS 00].

L'écoute sur haut-parleurs n'est pas sujette aux défauts d'extériorisation observés dans le cas d'une écoute au casque. Toutefois, le rendu spatial n'est optimal que pour un seul auditeur, placé en une position précise (le plus souvent au sommet d'un triangle équilatéral dont les deux haut-parleurs sont les autres sommets).

3.5. Les techniques stéréophoniques et les potentiomètres panoramiques

3.5.1. *Principe d'encodage et de restitution*

La stéréophonie, où « son en relief », permet de recréer une scène sonore spatialisée dans une ouverture formée par deux haut-parleurs. Le principe consiste à recréer les indices interauraux ITD et ILD aux oreilles de l'auditeur en introduisant des différences d'intensité (ΔI) et de temps (ΔT) entre les signaux qui alimentent les deux haut-parleurs [BAU 61, BERN 75, MAK 62]. Comme dans une situation transaurale, chaque oreille perçoit les sons provenant des deux haut-parleurs et il n'y a donc pas correspondance directe entre les indices interauraux (ITD, ILD) et le codage stéréo (ΔI, ΔT). Selon Bauer, dans le cas de sons basse fréquence à bande étroite, une différence d'intensité stéréophonique (ΔI) produit une différence de temps (IDT) aux oreilles, et inversement. Comme l'ITD basse fréquence est l'indice interaural dominant pour les sons large bande [WIG 92], la stéréophonie à été principalement développée selon une approche en variation d'intensité (ΔI). Bauer, Makita et Bernfeld ont déterminé les lois, appelées panoramiques d'intensité, qui

expriment la différence d'intensité ΔI permettant de produire une source virtuelle dans la direction ϕ_S à partir de deux haut-parleurs placés dans les directions $\pm \phi_{LP}$. Les lois obtenues sont très proches et les plus utilisées sont appelées lois des tangentes ou des sinus :

$$\Delta I = \frac{\tan\phi_{LP} - \tan\phi_S}{\tan\phi_{LP} - \tan\phi_S}$$

$$\Delta I = \frac{\sin\phi_{LP} - \sin\phi_S}{\sin\phi_{LP} - \sin\phi_S}$$

Un angle de 60° entre les haut-parleurs correspond au compromis optimal entre la largeur de la scène sonore et la robustesse des sources virtuelles créées.

Depuis la fin des années 1960, les techniques de panoramique ont été étendues aux systèmes multicanaux comportant plus de deux haut-parleurs placés sur un cercle. Le principe consiste à considérer séparément les paires adjacentes de haut-parleurs et d'appliquer directement à chaque paire les lois développées pour la stéréophonie à deux canaux. Récemment, les techniques de panoramiques ont été étendues à des configurations quelconques de haut-parleurs placés sur une sphère. Cette méthode appelée VBAP, positionne une source virtuelle en utilisant les trois haut-parleurs les plus proches, selon une extension de la loi des tangentes [PUL 97].

3.5.2. *Système d'enregistrement*

Un système d'enregistrement stéréophonique crée naturellement un encodage stéréophonique de la direction des sources auxquelles il est exposé. Dès 1930, Blumlein place deux microphones au point d'enregistrement et exploite leurs caractéristiques de directivité afin de créer des différences d'intensité ΔI entre les canaux qui dépendent de la direction des sources sonores [BLU 33]. Ces travaux sont à l'origine de la notion de couple stéréophonique coïncidant qui à été déclinée en de nombreuses techniques dont les plus connues sont les dénommées *Stéréosonic*, *M-S* et *XY*. Signalons que l'encodage ainsi obtenu est moins performant que les lois de panoramiques, car l'enregistrement doit s'accommoder des caractéristiques de directivité offertes par les microphones qui diffèrent nettement des lois de panoramique.

D'autres approches utilisent deux microphones espacés et réalisent un encodage en différence de temps de la direction des sources sonores en raison d'un temps de propagation différent pour que le son atteigne chacun des deux microphones. Ces travaux sont à l'origine de la notion de couple stéréophonique non coïncident qui a été déclinée en de nombreuses techniques dont les plus connues sont appelées *AB* et *ORTF*.

Depuis la fin des années 1960, les techniques d'enregistrement sont en cours d'extension aux systèmes multicanaux comportant plus de deux haut-parleurs placés sur un cercle. Toutefois cette extension est particulièrement complexe car, contrairement aux lois de panoramiques, les directivités offertes par les microphones ne permettent pas de décomposer le système multicanal en paires stéréophoniques juxtaposées et indépendantes.

Compte tenu de ces contraintes, Gerzon, Williams et Theile ont proposé des techniques fournissant des différences d'intensité ΔI et de temps ΔT optimales. Les plus connues sont appelées *Double M-S*, *Soundfield*, *INA*, *croix IRT*, *OCT* et *Multi-Microphone Array* [THE 01, WIL 00, WIL 01, WIL 02, WUT 01].

3.6. L'holophonie ou *Wave Field Synthesis*

Les systèmes holophoniques répondent au besoin de reproduction sonore de haute qualité dans une zone étendue. Ces systèmes sont basés sur une approche purement physique qui s'intéresse à enregistrer et reproduire l'ensemble des phénomènes ondulatoires qui constituent les sons, c'est-à-dire les champs acoustiques tridimensionnels. Les systèmes holophoniques captent un champ acoustique dans un espace et le reproduisent à l'identique dans un autre espace de sorte qu'un auditeur se déplaçant dans l'espace de restitution perçoive les mêmes effets que s'il se déplaçait dans l'espace de capture.

Par analogie avec l'holographie, l'holophonie est une méthode de reproduction d'un champ acoustique 3D dans un volume à partir de son enregistrement sur une surface [JES 73]. L'holophonie repose sur le principe de Huygens, énoncé en 1690 : le front d'onde rayonné par une source, appelée source primaire, se comporte comme une distribution de sources, appelées sources secondaires. Le champ produit par la distribution de sources secondaires se substitue parfaitement au champ produit par la source primaire. Ce principe a été quantifié par Kirchhoff et Helmoltz au XIX^{e} et XX^{e} siècle.

A la fin des années 1980, Berkhout propose de l'exploiter dans la conception de système audio de haute qualité [BER 88]. De Vries explore alors le potentiel de cette approche et le valide expérimentalement : c'est la naissance du *Wave Field Synthesis*. L'application directe de l'holophonie requiert un très grand nombre de transducteurs (microphones et haut-parleurs). Une grande partie des travaux sur le *Wave Field Synthesis* a consisté à réduire le nombre de transducteurs jusqu'à un nombre raisonnable (environ une centaine).

3.6.1. *Modèle de représentation*

Dans une représentation holophonique du champ acoustique, l'espace est décomposé en deux sous-espaces : Ω_1 contenant les sources primaires et Ω_2 ne contenant aucune source primaire et constituant l'espace de restitution. Le problème consiste à reproduire dans Ω_2 le champ acoustique engendré par les sources primaires en faisant uniquement intervenir une distribution de sources secondaires réparties sur la frontière S séparant les espaces Ω_1 et Ω_2.

Ce problème admet plusieurs solutions en fonction des hypothèses retenues, mais la plus répandue est l'intégrale de Kirchhoff-Helmoltz, exprimée dans le domaine fréquentiel. Cette solution suppose que l'espace soit vide d'obstacle (propagation en champ libre) et que le sous-espace de restitution Ω_2 soit entouré par la surface fermée S [NIC 99] :

$$P(\vec{x}, f) = \iint_S \left[\vec{\nabla} P_0(\vec{x}_0, f).\vec{n} - \frac{\vec{R}}{R}.\vec{n}(1 + jkR)\frac{P_0(\vec{x}_0, f)}{R} \right] \frac{e^{-jkR}}{4\pi R} dS$$

où :

- c est la célérité du son dans l'air (340 ms^{-1}) ;
- $\vec{x}$ désigne la position dans l'espace de restitution Ω_2 ;
- $\vec{x}_0$ désigne la position sur la surface S et correspond à la variable d'intégration ;
- $\vec{R} = \vec{x} - \vec{x}_0$ est un vecteur de norme R qui représente le trajet de $\vec{x}_0$ vers $\vec{x}$;
- $\vec{n}$ représente la normale unitaire à la surface S (figure 3.4) ;
- $P(\vec{x}, f)$ est le champ de pression restitué par les sources secondaires ;
- $P_0(\vec{x}, f)$ est le champ de pression induit par les sources primaires ;
- $k = \frac{2\pi f}{c}$ est le nombre d'onde.

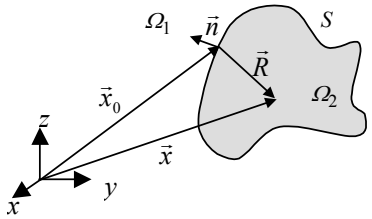


Figure 3.4. *Géométrie associée à l'intégrale de Kirchhoff*

3.6.2. *Principe d'enregistrement et de restitution*

L'intégrale de Kirchhoff-Helmoltz décrit explicitement un procédé physique d'enregistrement-restitution de champ acoustique tridimensionnel. L'enregistrement de l'onde primaire est obtenue par mesure sur la surface *S* de la pression $P_0(\vec{x}_0, f)$ et de la composante du gradient de pression $\vec{\nabla} P_0(\vec{x}_0, f).\vec{n}$ orienté selon la normale $\vec{n}$ à la surface *S*. La restitution met en œuvre une distribution de sources secondaires qui recouvre toute la surface *S*. En chaque point $\vec{x}_0$ de la surface *S*, la distribution est constituée d'une source de pression et d'une source de gradient de pression orientée suivant la normale $\vec{n}$ à la surface *S*. Les champs acoustiques produits par ces deux sources secondaires s'écrivent respectivement :

$$\frac{e^{-jkR}}{4\pi R}$$

et :

$$\frac{\vec{R}}{R}.\vec{n}\left(1+jkR\right)\frac{e^{-jkR}}{4\pi R^2}$$

Selon l'intégrale de Kirchhoff-Helmoltz, pour chaque point de la surface, les sources secondaires de pression sont pilotées par le gradient de pression de l'onde primaire et inversement. L'action simultanée de l'ensemble des sources secondaires permet de recréer l'onde primaire dans l'espace de restitution Ω_2 [NIC 99] Le principe de l'enregistrement et de restitution holophonique est illustré figure 3.5.

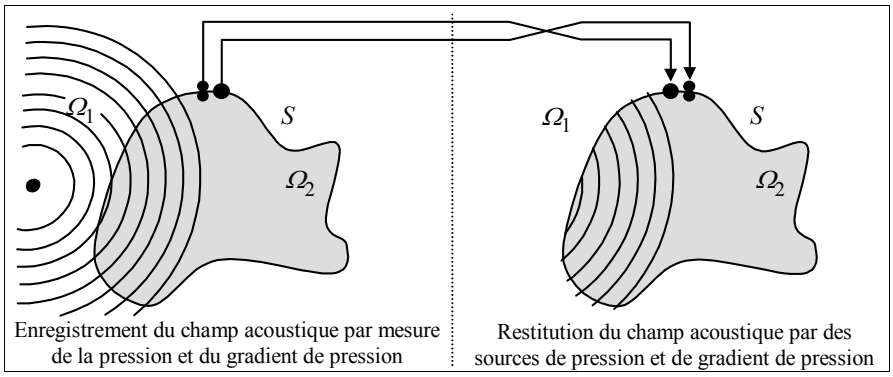


Figure 3.5. *Principe d'enregistrement et restitution holophonique*

L'intégrale de Kirchhoff-Helmoltz fonctionne également lorsque la surfaces S séparant les espaces Ω_1 et Ω_2 est un plan infini. Dans ce cas particulier, la reproduction du champ dans l'espace Ω_2 peut être assurée par un seul type de sources secondaires et l'intégrale de Kichoff-Helmoltz peut être simplifiée (intégrales de Raleigh).

3.6.3. *Echantillonnage, fréquence limite et* aliasing *spatial*

L'intégrale de Kirchhoff-Helmoltz exploite une distribution continue de microphones et de sources secondaires. Pour mettre en œuvre un système holophonique, il est nécessaire de substituer des distributions discrètes à ces distributions continues. L'enregistrement de l'onde primaire est alors réalisé par un ensemble de microphones ponctuels placés sur la surface S. De même, la reproduction de l'onde est assurée par un ensemble de haut-parleurs placés sur surface S. Ainsi, la mesure et la reconstruction du champ acoustique posent un problème d'échantillonnage de la surface S où le pas Δ entre les échantillons doit être adapté en fonction des variations de pression sur cette surface. Dans le cas d'une surface S plane, tout champ acoustique est convenablement échantillonné lorsque le pas Δ respecte le critère de Shannon :

$$\Delta < \frac{\lambda_{\min}}{2} = \frac{c}{2f_{\max}}, \text{ où } \lambda_{min}$$

est la plus courte longueur d'onde du champ acoustique.

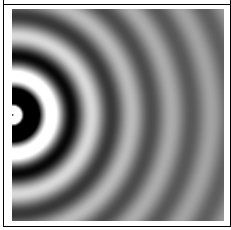
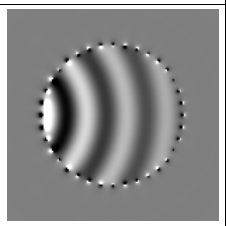
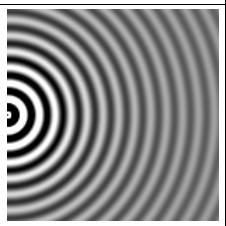
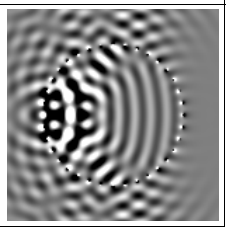
Sans repliement (f = 600 Hz)		Avec repliement (f = 1 500 Hz)	
Source primaire	Sources secondaires	Source primaire	Sources secondaires
			

Figure 3.6. *Illustration du phénomène de repliement de spectre spatial*

Ainsi, une reproduction holophonique jusqu'à une fréquence de 20 kHz nécessite de couvrir la surface avec des haut-parleurs ponctuels espacés de 8,5 mm, ce qui représente un nombre prohibitif. Dans le cadre d'une mise en œuvre pratique, le critère de Shannon ne peut pas être respecté dans les hautes fréquences. Il apparaît

alors un phénomène de repliement de spectre spatial au niveau de la surface S qui induit une erreur de reconstruction du champ acoustique dans l'espace Ω_2. La figure 3.6 illustre le phénomène de repliement de spectre dans le cas d'une surface sphérique d'1 m de rayon ou les sources secondaires sont placées tous les 10° en azimut et en élévation. La sphère est échantillonnée avec 614 points (1 228 sources secondaires) et permet une reproduction holophonique jusqu'à une fréquence de 974 Hz.

3.6.4. *Réduction du nombre de sources secondaires*

3.6.4.1. *Troncature du réseau de transducteurs*

Dans le processus de reconstruction du champ acoustique dans l'espace Ω_2, les sources secondaires ne contribuent pas avec la même importance. En raison des deux produits scalaires qui apparaissent dans l'intégrale de Kirchhoff-Helmoltz, les sources secondaires apportent une contribution maximale lorsque le trajet entre la source primaire, la source secondaire $\vec{x}_0$ et le point de reconstruction $\vec{x}$ est minimisé. Ainsi, si les sources primaires sont limitées à une certaine zone, il est possible de tronquer le réseau de sources secondaires tout en maintenant une qualité satisfaisante de reconstruction du champ acoustique. L'effet de la troncature est illustré sur la figure 3.7 dans le cas d'un réseau sphérique de sources secondaires. Il se traduit par une « fuite » du champ acoustique reproduit à l'extérieur de l'espace Ω_2.

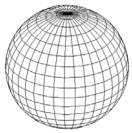
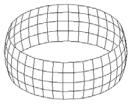
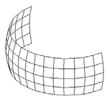
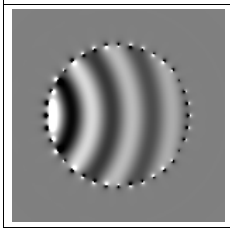
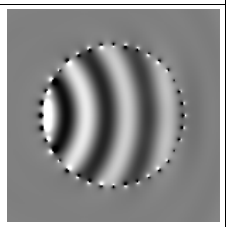
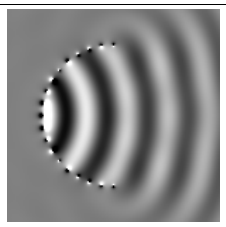
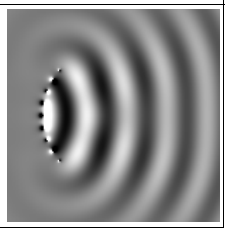
Réseau complet	**Réseau tronqué**		
1 228 sources	360 sources	190 sources	90 sources
			
			

Figure 3.7. *Influence de la troncature du réseau de sources secondaires*

Dans le cas où les sources primaires sont localisées uniquement dans le plan horizontal et la reproduction correcte du champ acoustique dans l'espace Ω_2 n'est souhaitée que pour le même plan horizontal, la surface S peut être convenablement approximée par une courbe C correspondant à l'intersection de la surface S et du plan horizontal. Cette simplification s'appuie sur le théorème de phase stationnaire. Grâce à cette approximation, la reproduction dans le plan peut être assurée avec moins de 100 transducteurs.

3.6.4.2. *Utilisation d'un seul type de sources secondaires*

L'intégrale de Kirchhoff-Helmoltz peut s'interpréter comme la superposition de deux champs acoustiques : l'un créé uniquement par les sources de pression et l'autre créé uniquement par les sources de gradient de pression. Il apparaît que ces deux champs agissent de manière constructive dans l'espace de restitution Ω_2 et destructive dans l'espace Ω_1 des sources primaires. Dans le cas particulier où la surface S est un plan infini, les contributions des sources de pression et de gradient de pression sont identiques dans l'espace Ω_2 et opposées dans l'espace Ω_1. Ainsi, la reconstruction du champ acoustique dans l'espace Ω_2 peut être parfaitement assuré par un seul type de sources (figure 3.8a). Dans le cas général, le champ peut être reproduit à l'intérieur de Ω_2 avec un seul type de sources, quelque soit la forme de la surface S. Cependant, les caractéristiques de directivité des microphones doivent être modifiées. La figure 3.8b illustre la reconstruction d'une onde avec des sources de gradient de pression, l'onde ayant été enregistrée avec des microphones cardioïdes (qui mesurent un mélange de pression et de gradient de pression).

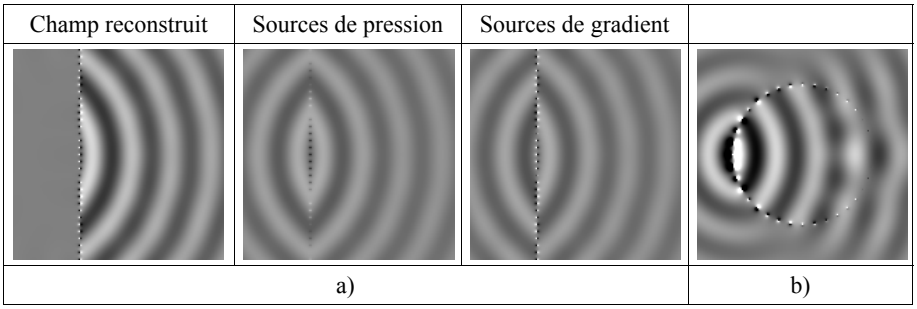


Figure 3.8. *Reconstruction du champ avec une surface plane. Contribution des sources de pression et de gradient de pression*

3.6.4.3. *Enregistrement et restitution sur des surfaces différentes*

Le principe de l'holophonie impose que la géométrie du réseau de capture et du réseau de sources secondaires soit identique, ce qui constitue une limite importante. Afin de relâcher cette contrainte, de Vries a introduit le concept d'extrapolation de

champ acoustique. L'intégrale de Kirchhoff-Helmoltz peut être interprétée comme un opérateur d'extrapolation depuis la surface S vers toute autre surface S_2 incluse dans l'espace Ω_2. En effet, la connaissance de la pression et du gradient de pression sur la surface S permet de déterminer le champ acoustique dans l'espace Ω_2 et donc la pression et le gradient de pression le long de la surface S_2. Ainsi, une reproduction holophonique de l'onde primaire depuis la surface S_2 est possible pour l'espace situé à l'intérieur de la surface S_2. En pratique, l'intégrale de Kirchhoff-Helmoltz est discrétisée et l'opérateur d'extrapolation du champ prend la forme d'une matrice de filtres $\boldsymbol{W}(f)^+$ qui relie chaque microphone sur S à chaque haut-parleur placé sur S_2. De même, un opérateur d'extrapolation inverse $\boldsymbol{W}(f)^-$ permet de reproduire le champ acoustique à partir d'une surface S_1 appartenant à l'espace Ω_1 et enveloppant la surface S. Ainsi, en combinant les opérateurs d'extrapolation direct $\boldsymbol{W}(f)^+$ et inverse $\boldsymbol{W}(f)^-$, le champ mesuré sur une surface S quelconque peut être reproduit à partir d'une autre surface quelconque [BER 99, HUL 01].

3.7. Les systèmes Ambisonic

A la fin des années 1960, les chercheurs Cooper/Shiga, et Gerzon développent indépendamment un modèle de représentation de l'environnement sonore en « tant que tel », indépendante des moyens d'acquisition et de restitution. Selon cette approche, l'environnement sonore est assimilé à une distribution angulaire de sources sonores autour d'un point correspondant à la position de l'auditeur. De la même manière qu'un télescope est sensible à la direction des étoiles avec une certaine netteté, les systèmes Ambisonic sont sensibles à la direction des sources sonores qui se distinguent avec une certaine précision, le système assurant une précision identique pour toutes les directions.

A partir du milieu des années 1990, cette approche rejoint l'acoustique fondamentale : il en résulte un puissant outil permettant le contrôle des champs acoustiques tridimensionnels dans une large zone de l'espace. Aujourd'hui, Ambisonic répond, comme l'holophonie, au besoin de reproduction sonore de haute qualité dans une zone étendue.

3.7.1. *Modèle de représentation initial : les harmoniques sphériques*

Le formalisme Ambisonic utilise un système de coordonnées sphérique conventionnel. Au travers d'un système Ambisonic, les signaux émis par la distribution angulaire des sources forment une fonction de directivité $d(\theta,\phi,t)$ qui dépend de la direction (θ,ϕ) et qui évolue au cours du temps t.

Afin de garantir une précision indépendante de la direction, la directivité est décrite sur une base orthonormée de fonctions de directivité appelées harmoniques sphériques. Les harmoniques sphériques sont aux fonctions de directivité ce que les harmoniques (fonctions $e^{j2\pi ft/T}$ ou $\sin(2\pi ft/T)$ | $\cos(2\pi ft/T)$) sont aux signaux 1D périodiques (de période T). Ainsi, les harmoniques sphériques permettent d'obtenir une représentation spectrale des fonctions de directivité [GER 72]. Comme les harmoniques, qui sont ordonnées selon leur fréquence f ($f = 0$ (composante continue), $f = 1$ (fondamental), $f = 2\ldots$), les harmoniques sphériques forment un ensemble hiérarchique classé selon « l'ordre », noté l, qui correspond à la finesse des lobes (figure 3.9). Pour un ordre l donné, il existe $2l + 1$ harmoniques sphériques qui sont ordonnées en fonction de leur terme m allant de $-l$ à l. Les harmoniques sphériques notées $y_l^m(\theta,\phi)$ sont définies par l'expression [MOO 01a] :

$$y_l^m(\theta,\phi) = \begin{cases} \frac{1}{\sqrt{\pi}} P_l^{|m|}(\cos\theta)\cos(m\phi) & \text{pour } m > 0 \\ \frac{1}{\sqrt{2\pi}} P_l^{0}(\cos\theta) & \text{pour } m = 0 \\ \frac{1}{\sqrt{\pi}} P_l^{|m|}(\cos\theta)\sin(m\phi) & \text{pour } m < 0 \end{cases}$$

Dans cette équation, les $P_l^m(x)$ sont les fonctions de Legendre associées définies par :

$$P_l^m(x) = \sqrt{\frac{2l+1}{2}}\sqrt{\frac{(l-m)!}{(l+m)!}}\left(1-x^2\right)^{m/2}\frac{d^m}{dx^m}p_l(x)$$

avec $P_l(x)$ les polynômes de Legendre, définis par :

$$P_l(x) = \frac{1}{2^l l!}\frac{d^l}{dx^l}\left(x^2+1\right)^l$$

Jusqu'à l'ordre un, elles prennent des expressions analytiques simples :

$$y_0^0(\theta,\phi) = \frac{1}{\sqrt{4\pi}}$$

$$y_1^{-1}(\theta,\phi) = \frac{\sqrt{3}}{\sqrt{4\pi}}\sin\theta\sin\phi$$

$$y_0^0(\theta,\phi) = \frac{\sqrt{3}}{\sqrt{4\pi}}\cos\theta$$

$$y_1^1(\theta,\phi) = \frac{\sqrt{3}}{\sqrt{4\pi}}\sin\theta\cos\phi$$

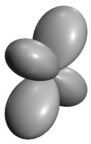
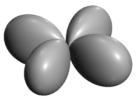
$y_0^0(\theta,\phi)$	$y_1^0(\theta,\phi)$	$y_2^0(\theta,\phi)$	$y_2^1(\theta,\phi)$	$y_2^2(\theta,\phi)$
				

Figure 3.9. *Quelques harmoniques sphériques*

D'une manière générale, les harmoniques sphériques offrent des propriétés mathématiques analogues aux fonctions exponentielles complexes (ou sinus/cosinus), mais reposent sur un formalisme plus lourd. Notamment, toute fonction de directivité continue $d(\theta,\phi,t)$ peut être décomposée en une série discrète de composantes élémentaires $d_{l,m}(t)$ selon une formule d'analyse appelée « transformée de Fourier sphérique » comparable à la transformée de Fourier habituelle :

$$d_{l,m}(t) = \int_{\theta=0}^{\pi}\int_{\phi=0}^{2\pi} d(\theta,\phi,t)\, y_l^m(\theta,\phi)\sin\theta\, d\theta\, d\phi$$

Chaque composante $d_{l,m}(t)$ est un signal temporel tout à fait standard, ce qui facilite la transmission mais aussi le traitement. De même, une fonction de directivité $d(\theta,\phi,t)$ se reconstruit à partir de ses composantes $d_{l,m}(t)$ selon une formule de synthèse appelée « transformée de Fourier sphérique inverse » comparable à la transformée de Fourier inverse habituelle :

$$d(\theta,\phi,t) = \sum_{l=0}^{\infty}\sum_{m=-l}^{l} d_{l,m}(t)\, y_l^m(\theta,\phi)$$

Les harmoniques sphériques offrent un fondement scientifique permettant en théorie une représentation de la direction des sources aussi précise que l'on souhaite. L'orthonormalité et la hiérarchie des harmoniques sphériques offre une grande compatibilité entre représentations de précision différente : la précision est réduite

par troncature (à un ordre donné L) ou augmentée par ajout des composantes manquantes [GER 72].

Ambisonic a longtemps été limité à une représentation d'ordre un, et les composantes $d_{0,0}(t)$, $d_{1,-1}(t)$, $d_{1,0}(t)$, $d_{1,1}(t)$ sont habituellement nommées W, Y, Z, X. Elles forment un ensemble de quatre signaux appelé « Format-B » et s'appréhendent de manière intuitive. La composante W correspond à la moyenne pour toutes les directions des signaux émis par les sources et constitue un signal de référence qui véhicule les signaux émis par les sources, sans spatialisation. On parle de composante omnidirectionnelle. Par un raisonnement similaire, on montre que les composantes X, Y et Z forment un codage explicite de la direction des sources. Malheureusement, les composantes d'ordre supérieur à 1 ne s'appréhendent pas de manière aussi intuitive.

3.7.2. *Généralisation du modèle de représentation : les fonctions de Fourier-Bessel*

Depuis le milieu des années 1990, le modèle a été rapproché de l'acoustique fondamentale qui offre un modèle complet des champs acoustiques linéaires. Ce modèle s'intéresse à la modélisation du champ acoustique dans la zone d'écoute plutôt qu'à la distribution des sources sonores observées depuis un point [BAM 95, DAN 98, DAN 03, NIC 98, NIC 99, POL 00].

Cette nouvelle approche repose sur les séries de Fourier-Bessel, une théorie développée vers 1870. Un champ acoustique est connu si l'on définit en tout point (r,θ,ϕ) et à chaque instant t la pression acoustique notée $p(r,\theta,\phi,t)$. La représentation dans le domaine fréquentiel, notée $P(r,\theta,\phi,f)$, s'obtient par transformée de Fourier de $p(r,\theta,\phi,t)$.

Dans une zone vide de sources sonores et vide d'obstacles, les fonctions de Fourier-Bessel sont les solutions générales de l'équation des ondes et constituent une base qui engendre tous les champs acoustiques produits par des sources sonores situées à l'extérieur de cette zone [BRU 98]. Tout champ acoustique tridimensionnel s'exprime donc par une combinaison linéaire des fonctions de Fourier-Bessel, selon l'expression de la transformée de Fourier-Bessel inverse qui s'exprime par :

$$P(r,\theta,\phi,f) = 4\pi \sum_{l=0}^{\infty} \sum_{m=-l}^{l} P_{l,m}(f)\, j^l\, j_l(kr)\, y_l^m(\theta,\phi)$$

Dans cette équation, les termes $P_{l,m}(f)$ sont, par définition, les coefficients de Fourier-Bessel du champ $p(r,\theta,\phi,t)$. Le terme $j^l\, j_l\left(kr\right) y_l^m\left(\theta,\phi\right)$ correspond à la fonction de Fourier-Bessel d'ordre l et de terme m composée de la fonction de

Bessel sphérique de première espèce d'ordre l $j_l(x)$ et de l'harmonique sphérique $y_l^m(\theta,\phi)$ définie précédemment. Dans cette expression $k=\frac{2\pi f}{c}$ est le nombre d'ondes et c est la célérité du son dans l'air (340 ms^{-1}).

Les coefficients de Fourier-Bessel $P_{l,m}(f)$ s'expriment aussi dans le domaine temporel par les coefficients $p_{l,m}(t)$, le passage s'effectue par transformée de Fourier.

Les fonctions de Fourier-Bessel constituent une base hiérarchique de champs acoustiques élémentaires qui permettent d'engendrer tout champ acoustique et permettent d'obtenir une représentation spectrale des champs acoustiques. Les fonctions de Fourier-Bessel constituent un outil de description plus complet que les harmoniques sphériques, car elles permettent de représenter le comportement radial du champ acoustique (comportement selon la dimension r). Les premières fonctions de Fourier-Bessel sont illustrées par la figure 3.10.

La zone de l'espace autour de l'origine O dans laquelle le champ acoustique est convenablement représenté est proportionnelle à la longueur d'onde et à l'ordre L de troncature de la série. Ainsi, l'augmentation du nombre de coefficients permet d'accroître la zone dans laquelle le champ acoustique est convenablement représenté [BV95, DAN 03] (figure 3.11).

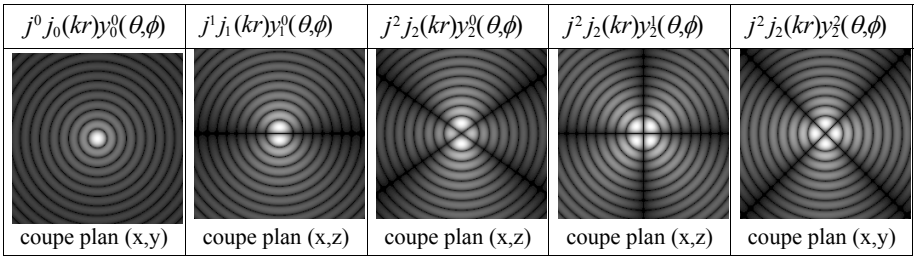


Figure 3.10. *Quelques fonctions de Fourier-Bessel. Chaque image correspond au module des fonctions de Fourrier-Bessel évaluées selon le plan spécifié*

Toutefois, les deux modélisations sont liées : sous l'hypothèse que les sources sonores sont toutes placées à l'infini, les coefficients de Fourrier-Bessel du champ acoustique correspondent aux composantes d'harmoniques sphériques de la distribution des sources sonores. Ainsi, la modélisation par fonction de Fourrier-Bessel peut s'envisager soit comme une généralisation de l'approche initiale d'Ambisonic, issue elle-même d'une généralisation de l'approche de Blumlein, soit comme une application de l'acoustique fondamentale du XIX^{e} siècle.

3.7.3. *Principe d'enregistrement*

3.7.3.1. *Enregistrement à l'ordre un*

Un enregistrement Ambisonic correspond à la mise en œuvre de la formule de la transformée de Fourier sphérique en utilisant un ensemble de capteurs placés au point d'observation dans lequel chaque capteur présente une directivité correspondant à une harmonique sphérique particulière. On entend par directivité d'un capteur sa sensibilité au signal émis par une source en fonction de la direction de la source. Selon cette approche, les composantes *W*, *X*, *Y* et *Z* du champ sont respectivement obtenues par mesure à l'origine de la pression et de la vitesse (ou vélocité particulaire) selon les axes *Ox*, *Oy* et *Oz*. Cette approche souffre de deux principales limitations :

– l'absence de capteurs dont la directivité est d'ordre supérieur à un,

– l'impossibilité de positionner l'ensemble des capteurs au point d'observation.

En 1975, Craven et Gerzon parviennent à surmonter en partie la seconde limitation en proposant un système de prise de son simulant le fonctionnement d'un système coïncident dans lequel l'ensemble des capteurs serait placé en un même point. Cette technique ne permet pas de recomposer des harmoniques sphériques d'un ordre supérieur à la directivité intrinsèque des capteurs et fournit donc une représentation limitée à l'ordre un (en raison de l'orthogonalité des harmoniques sphériques). Cette technique a été appliquée avec succès à un réseau tétraédrique de microphones cardioïdes et est connue sous le nom de microphone *Soundfield* [FAR 79].

Alternativement, la mesure de la pression et de la vitesse à l'origine peut être obtenue à l'aide d'un réseau tétraédrique de microphones omnidirectionnels. Ce système exploite la distance intercapteur pour estimer la vitesse selon ses trois composantes [ELK].

3.7.3.2. *Enregistrement aux ordres supérieurs à un*

Dans son approche initiale basée uniquement sur les harmoniques sphériques, la captation de composantes d'ordre supérieures à un impose l'utilisation de capteurs qui présentent une directivité intrinsèque faisant intervenir des composantes d'ordre supérieur à un (propriété d'orthogonalité des harmoniques sphériques). Ainsi, dès le début des années 1970, Gerzon s'est intéressé aux réseaux de microphones permettant de recréer des microphones directifs selon des techniques de formation de voie inspirées des antennes radar [GER 7X]. Malheureusement, de tels systèmes sont adaptés pour fonctionner sur une bande de fréquence étroite et requièrent un très grand nombre de microphones élémentaires pour obtenir un directivité pointue sur une large bande de fréquence [MOO 01b].

D'autres approches basées sur les séries de Fourier-Bessel exploitent le comportement radial du champ acoustique et s'affranchissent des principales contraintes de la première approche : il n'est plus nécessaire de concentrer l'ensemble des capteurs en un même point et les capteurs ne doivent pas fournir directement des directivités d'ordre supérieur à un. La démarche consiste :

– à réaliser un échantillonnage spatial du champ de pression à la surface d'une sphère fictive avec un ensemble de capteurs répartis sur cette sphère ;

– à calculer la transformée de Fourier sphérique du champ sur la sphère, à partir des échantillons ;

– à déduire les coefficients de Fourier-Bessel du champ acoustique en appliquant aux coefficients une égalisation propre à chaque ordre.

En exploitant un réseau circulaire de microphones, Poletti a proposé une approche permettant d'enregistrer les champs acoustiques 2D décrits par les fonctions de Fourier-Bessel cylindriques [POL 00]. Elko, Mayer et Kubli proposent une approche permettant d'enregistrer les champs acoustiques 3D en exploitant la diffraction du champ acoustique à la surface d'une sphère rigide [ELK 03].

Une autre approche repose sur une généralisation du concept d'échantillonnage spatial à tout ensemble de capteurs de directivité quelconque répartis et orientés de manière quelconque dans l'espace. Si le champ acoustique dans lequel sont plongés les capteurs est connu, il est possible de déterminer les signaux délivrés par les capteurs grâce à une relation linéaire. Il existe ainsi une matrice **B** d'échantillonnage spatial qui fournit les signaux des capteurs à partir des coefficients de Fourier-Bessel du champ acoustique auquel ils sont exposés. Or, le principe du microphone est précisément l'inverse : on dispose des signaux et l'on souhaite estimer le champ acoustique initial. Donc pour réaliser un microphone, il suffit d'inverser la matrice d'échantillonnage **B** à l'aide de techniques d'inverse généralisés [LAB 03].

3.7.4. *Synthèse*

Le formalisme d'Ambisonic permet d'encoder une source ponctuelle placée à l'infini dans la direction (θ_0,ϕ_0) et émettant un signal s(t). Selon l'approche angulaire initiale, l'encodage correspond à l'échantillonnage angulaire de la base d'harmoniques sphériques dans la direction (θ_0,ϕ_0), alors que selon l'approche champ, l'encodage correspond à la décomposition en série de Fourier-Bessel d'une onde plane portant le signal s(t) et provenant de la direction (θ_0,ϕ_0). Les deux représentations sont équivalentes lorsque la source est placée à l'infini. L'encodage est obtenu au moyen de la relation suivante :

$$d_{l,m}(t) = p_{l,m}(t) = y_l^m(\theta_0,\phi_0)\,s(t)$$

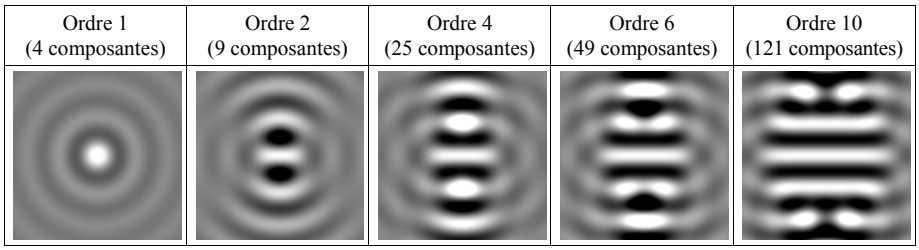


Figure 3.11. *Champ acoustique correspondant à la représentation d'une onde plane de direction (0,0). Chaque image correspond à une coupe dans le plan (x,z)*

L'approche Fourier-Bessel permet d'encoder le rayonnement sphérique d'une source sonore placée à une distance finie r_0 de l'origine O [DAN 03, LAB 03]. Une source proche de l'origine se distingue d'une source placée à l'infini par des modifications d'amplitude (amplification) et de phase affectant les coefficients de Fourier-Bessel.

3.7.5. *Traitement*

Lorsque l'on souhaite effectuer un traitement sur un signal, il est souvent plus efficace de manipuler sa représentation fréquentielle que d'agir directement sur sa forme d'onde temporelle.

De même la représentation spectrale sur laquelle repose Ambisonic autorise de nombreux traitements spatiaux qui s'expriment sous la forme de matrices recombinant les composantes spectrales et influençant la spatialisation :

$$\begin{bmatrix} d'_{0,0}(t) \\ \vdots \\ d'_{l,m}(t) \\ \vdots \\ d'_{L,M}(t) \end{bmatrix} = \begin{bmatrix} \text{Matrice de} \\ \text{traitement} \\ (L+1)^2 \times (L+1)^2 \end{bmatrix} \bullet \begin{bmatrix} d_{0,0}(t) \\ \vdots \\ d_{l,m}(t) \\ \vdots \\ d_{L,M}(t) \end{bmatrix}$$

Gerzon et Malham ont proposé un certain nombre de transformations s'appliquant à une représentation d'ordre un, notamment la rotation globale de la directivité $d(\theta,\phi,t)$ autour de chacun des axes Ox, Oy et Oz du repère cartésien et une distorsion de perspective (*forward dominance*). Le tableau 3.1 donne les matrices de transformation correspondantes.

Rotation de α selon Ox	Rotation de β selon Oy	Rotation γ selon Oz	*Forward dominance*
$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos\alpha & -\sin\alpha \\ 0 & 0 & \sin\alpha & \cos\alpha \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & 0 & -\sin\beta \\ 0 & 0 & 1 & 0 \\ 0 & \sin\beta & 0 & \cos\beta \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma & 0 \\ 0 & \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & \mu & 0 & 0 \\ \mu & 1 & 0 & 0 \\ 0 & 0 & \sqrt{1-\mu^2} & 0 \\ 0 & 0 & 0 & \sqrt{1-\mu^2} \end{bmatrix}$

Tableau 3.1. *Matrices de traitement de champ acoustique à l'ordre 1*

Certaines opérations ont été étendues pour les ordres supérieurs, en particulier, la rotation pour une description d'ordre supérieur à 1. En général, les coefficients des matrices de transformation agissant sur des représentations d'ordre supérieur à 1 s'obtiennent avec des relations assez lourdes.

3.7.6. *Principe de restitution*

Une restitution Ambisonic est obtenue en plaçant un ensemble de N haut-parleurs autour de l'origine O correspondant au point d'écoute. Tout l'enjeu consiste à déterminer les signaux destinés à alimenter chacun des haut-parleurs à partir de la représentation de l'environnement sonore selon une opération appelée *décodage*.

3.7.6.1. *Décodage physique*

Une première famille de décodeurs repose sur une approche purement physique. Chaque haut-parleur n du système de restitution constitue une source sonore dans la direction (θ_n,ϕ_n) émettant un signal $u_n(t)$ et dont l'encodage directionnel en composantes Ambisonic correspond à un échantillonnage des harmoniques sphériques pour cette direction. Ainsi, les composantes Ambisonic $d_{l,m}(t)$ sont obtenues en fonction des signaux $u_n(t)$ grâce à une matrice **M** d'encodage qui prend la forme suivante :

$$\begin{bmatrix} d_{0,0}(t) \\ \\ d_{l,m}(t) \\ \\ d_{L,M}(t) \end{bmatrix} = \begin{bmatrix} M_{0,0,1} & \cdots & M_{0,0,n} & \cdots & M_{0,0,N} \\ \vdots & & \vdots & & \vdots \\ M_{l,m,1} & \cdots & M_{l,m,n} & \cdots & M_{l,m,N} \\ \vdots & & \vdots & & \vdots \\ M_{L,M,1} & \cdots & M_{L,M,n} & \cdots & M_{L,M,N} \end{bmatrix} \bullet \begin{bmatrix} u_1(t) \\ \vdots \\ u_n(t) \\ \vdots \\ u_N(t) \end{bmatrix}$$

avec $M_{l,m,n} = y_l^m(\theta_n,\phi_n)$.

La problématique du décodage est exactement inverse : les composantes à restituer sont connues et l'on souhaite déterminer les signaux des haut-parleurs afin de les reproduire. Ainsi, le décodage s'exprime par une matrice **D** obtenue en inversant la précédente matrice d'encodage selon la relation, [NIC 98, NIC 99] :

– $\mathbf{D} = \mathbf{M}^t.(\mathbf{M}.\mathbf{M}^t)^{-1}$ s'il y a plus de haut-parleurs que de composantes ($N > (L + 1)^2$) ;

– $\mathbf{D} = (\mathbf{M}.\mathbf{M}^t)^{-1}.\mathbf{M}^t$ s'il y a moins de haut-parleurs que de composantes ($N < (L + 1)^2$).

Ce type de décodage est particulièrement adapté à des configurations régulières dans lesquelles les haut-parleurs sont répartis régulièrement sur une sphère fictive. Malheureusement, il n'existe que cinq dispositions régulières sur la sphère (sommets des cinq polyèdres réguliers) et les performances obtenues avec des configurations non régulières sont dégradées.

L'approche Fourrier-Bessel permet de prendre en compte le rayonnement sphérique des sources sonores dans le cas où les haut-parleurs sont placés à proximité de l'auditeur où le modèle d'onde plane (implicite avec l'approche angulaire) n'est plus vérifié.

3.7.6.2. *Décodeurs perceptifs*

Une autre famille de décodeurs exploite les propriétés du système auditif humain pour réaliser une optimisation perceptive de la localisation. Gerzon utilise le vecteur énergie $\vec{E}$ comme double prédicteur de la localisation haute fréquence et de la localisation en position d'écoute excentrée par rapport à l'origine *O*. Les techniques de décodage physique assurant naturellement une orientation correcte du vecteur énergie, l'optimisation porte sur la maximisation de sa norme : c'est l'optimisation *max*-r_E [DRP 98, GER 92].

Une autre approche perceptive, proposée par Malham, considère le décodage comme un ensemble de microphones virtuels directifs pointant dans la direction des haut-parleurs et recherche les solutions pour lesquelles leurs directivités ne présentent aucun lobe secondaire. Cette approche, appelée *in-phase*, permet d'accroître la zone de restitution [MAL 92].

De nombreux décodeurs reposent sur une approche hybride physique/perceptive, où le décodage est physique pour les basses fréquences et éventuellement perceptif pour les hautes fréquences, en fonction de la répartition des auditeurs :

– à l'origine → décodage *physique* ;

– au voisinage de l'origine → décodage *max*-r_E ;

– dans une zone étendue → décodage *in-phase*.

La transition entre les deux méthodes met en œuvre des filtres appelées *shelving filters* en raison de leur réponse caractéristique.

3.7.7. *Ambisonic et holophonie*

Les systèmes Ambisonic et holophoniques ont pour vocation l'enregistrement et la reproduction des champs acoustiques tridimensionnels dans une zone de l'espace. Nicol et Emerit de France Telecom R&D ont montré qu'il s'agit de représentations équivalentes lorsque le champ est reproduit par une surface sphérique recouverte de sources d'onde plane [NIC 99].

La caractérisation exhaustive du champ acoustique dans une large zone ne peut être réalisée en pratique car elle nécessite une trop grande quantité d'information. Les approches holophoniques et Ambisonic se différencient principalement dans l'approche retenue pour réduire la quantité d'information [NIC 99] :

– l'holophonie échantillonne la surface qui enveloppe la zone de restitution. Au-delà d'une fréquence limitée, la reproduction devient incorrecte dans toute la zone ;

– Ambisonic tronque la représentation spectrale du champ acoustique. En contrepartie, la zone de reproduction diminue lorsque la fréquence augmente.

Le choix de l'un des deux systèmes dépend étroitement de l'application envisagée. Toutefois, les systèmes Ambisonic présentent certains avantages qu'il convient de souligner. Nicol et Emerit ont mis en évidence qu'Ambisonic constitue un encodage plus efficace de l'information spatiale [NIC 98, NIC 99]. Les propriétés hiérarchiques de la représentation Ambisonic permettent un ajustement immédiat de la qualité de représentation et la quantité de données transmises. Enfin, Ambisonic offre une représentation adaptée pour réaliser des traitements sur le champ acoustique.

3.8. Conclusion

Les techniques de spatialisation des sons se sont développées avec le désir des compositeurs de manipuler le champ acoustique entourant les auditeurs en situation de concert. Elles dépassent aujourd'hui le cadre du concert et sont parfois très largement diffusées, par exemple sur les cartes sons pour PC, avec pour principale application les jeux vidéos. On parle alors de « sons 3D ».

Les approches décrites dans ce chapitre répondent à différents besoins. Dans le tableau 3.2, nous comparons les caractéristiques de chacune sur trois facteurs. Le type de transducteurs utilisé pour la diffusion sonore différencie les techniques pour le casque des techniques pour haut-parleurs, et peut avoir une influence sur le coût financier et les difficultés logistiques de l'installation lorsque, comme dans le cas de

l'holophonie, de nombreux haut-parleurs sont nécessaires. Toutes les techniques décrites comportent une étape d'encodage spatial qui peut être réalisé par synthèse à partir de sons monophoniques. Certaines offrent l'avantage supplémentaire de pouvoir spatialiser des scènes sonores naturelles lorsqu'il existe une technique de prise de son équivalente. Enfin, il est important de prendre en compte la taille de la zone d'écoute dans laquelle la restitution spatiale sera satisfaisante. L'holophonie et Ambisonic peuvent reproduire l'image sur une zone d'écoute étendue, mais au prix d'un plus grand nombre de transducteurs. Toutes les autres ne sont optimales que pour un point d'écoute précis (le *sweet spot*) et doivent alors être comparées sur la rapidité avec laquelle l'image spatiale se dégrade lorsque l'on s'éloigne de ce point d'écoute.

	Type de transducteurs pour la diffusion	**Prise de son équivalente**	**Taille de la zone d'écoute**
Techniques binaurales	Casque d'écoute	Oui	Ecoute individuelle
Techniques transaurales	Paire de haut-parleurs	Oui	Ecoute individuelle
Holophonie	Un grand nombre de haut-parleurs	Oui	Large zone d'écoute jusqu'à une fréquence limite
Ambisonic	A partir de quatre haut-parleurs	Oui	Zone d'écoute dont la taille diminue lorsque la fréquence augmente
Potentiomètre panoramique d'intensité	Paire (ou triplet) de haut-parleurs	Non	Zone d'écoute réduite

Tableau 3.2. *Xxxxxxxxxx Légende ?*

3.9. Bibliographie

[BAU 61] BAUER B., « Phasor Analysis of Some Stereophonic Phenomena », *JASA*, vol. 33, n° 11, novembre 1961.

[BER 88] BERKHOUT A.J., « A Holographic approach to acoustic control », *JAES*, vol. 36, p. 977-995, décembre 1988.

[BER 99] BERKHOUT A.J., DE VRIES D., BAAN J., VAN DER OETELAAR B.W., « A wave field extrapolation approach to acoustical modeling in enclosed spaces », *JASA*, 105 (3), mars 1999.

[BERN 75] BERNFELD B., « Simple Equations for Multichannel Stereophonic Sound Localization », *JAES*, vol. 23, n° 7, septembre 1975.

[BAM 95] BAMFORD J.S., VANDERKOOY J., « Ambisonic Sound For Us », *99th AES Convention*, 4138, octobre 1995.

[BURA 91] BURANDT U., POSSELT C., AMBROZIS S., HOSENFELD M., KNAUFF V., « Anthropomorphic contribution to standardising mannequins for artificial-head microphones and to measuring headphones and ear protectors », *Applied Ergonomics*, 1991.

[BRU 98] BRUNEAU M., *Manuel d'acoustique fondamentale*, Hermès, Paris, 1998.

[BUR 75] BURKHARD M.D., SACHS R.M., « Anthropomorphic manikin for acoustic research », J. *Acoust. Soc. of Am*., 58(1), p. 214-222, 1975.

[CHO 71] CHOWNING J., « The Simulation of Moving Sound Sources », *Journal of the Audio Eng. Soc.*, vol. 19, n° 1, janvier 1971.

[COO 89] COOPER D.H., BAUCK J.L., « Prospects for transaural recording », *Journal of the Audio Eng. Soc*., vol. 37, n° 1/2, janvier 1989.

[CHE 96] CHEN J., VAN VEEN B.P., HECOX K.E., Methods and apparatus for producing directional sound, United States patents (5500900), mars 1996.

[DHO 88] DHOMONT F. (DIR.), *L'espace du son*, Ohain (Belgique) : Musique et Recherche, 1988.

[DAN 98] DANIEL J., RAULT J.-B., POLACK J.-D., « Ambisonic Encoding of Other Audio Formats for Multiple Listening Conditions », *105th AES Convention*, 4795, septembre 1998.

[DAN 03] DANIEL J., « Spatial Sound Encoding Including Near Field Effect : Introducing Distance Coding Filters and a Viable New Ambisonic Format. », *AES 23rd International Conference*, mai 2003.

[ELK] ELKO G.W., « A Simple First-Order differential microphone », Bell Labs, Lucent Technologies.

[ELK 03] ELKO G.W., KUBLI R.A., MEYER J., Audio System based on at least second-order eigenbeams, Demande de brevet d'invention US 2003/0147539, août 2003.

[FAR 79] FARRAR K., « Soundfield microphone – 1. Design and development of microphone and control unit », *Wireless World*, p. 48-50, octobre 1979.

[GER 73] GERZON M., « Periphony: With-Height Sound Reproduction », *JAES*, vol. 21, n° 1, p. 2-10, janvier 1973.

[GER 7X] GERZON M., Ultra-Directional Microphones : Part 1, Part 2 et Part 3.

[GER 92] GERZON M., « Psychoacoustic Decoder for Multispeaker Stereo and Surround Sound », *93rd Convention AES*, 3406, octobre 1992.

[HAR 99] HARTUNG K., BRAASCH J., STERBING S., « Comparison of different methods for the interpolation of HRTF », *16th conference of the Audio Eng. Soc*., 1999.

[HUL 01] HULSEBOS E., DE VRIES D., BOURDILLAT E., « Improved microphone array configuration for auralisation of sound fields by Wave Field Synthesis », *110th AES Convention*, mai 2001.

[JES 73] JESSEL M., *Acoustique théorique, propagation et holophonie*, Masson, Paris, 1973.

[JOS 00] JOST A., JOT J.-M., « Transaural 3-D Audio with user-controlled calibration », *Proceedings of the Conference on Dig. Audio Effects*, Veronq, 2000.

[JOT 95] JOT J.-M., LARCHER V., WARUSFEL O., « Digital Signal Processing issues in the context of Binaural and Transaural Stereophony », *98th AES Convention*, 3980(I6), février 1995.

[JOT 92] JOT J.-M., Etude et réalisation d'un spatialisateur de sons par modèles physique et perceptif, Thèse de Doctorat, Ecole Nationale Supérieure des Télécommunications, septembre 1992.

[JUL 94] JULLIEN J.-P., WARUSFEL O., « Technologies et perception auditive de l'Espace », *Cahiers de l'Ircam*, 1994.

[JOT 95] JOT J.-M., WARUSFEL O., « Le Spatialisateur », *Actes du colloque « Le son et l'espace ». Rencontres musicales pluridisciplinaires Informatique et Musique. GRAME*, Musiques en Scènes, Lyon, 1995.

[JOT 98] JOT J.-M., WARDLE S., LARCHER V., « Approaches to binaural synthesis », *105th AES Convention*, 4861(K4), 1998.

[LAR 95] LARCHER V., Paramétrisation des fonctions de transfert binaurales, Mémoire Ingénieur de l'Ecole nationale supérieure des télécommunications, 1995.

[LBM 03] LABORIE A., BRUNO R., MONTOYA S., « A New Comprehensive Approach of Surround Sound Recording », *114th AES Convention*, 5717, mars 2003.

[LAR 97] LARCHER V., JOT J.-M., « Techniques d'interpolation de filtres audio-numériques. Application à la reproduction spatiale des sons sur écouteurs », *4e congrès français d'acoustique*, 1997.

[LAR 98] LARCHER V., JOT J.-M., VANDERNOOT G., « Equalization methods in Binaural Technology », *105th convention of the Audio Eng Soc*., #4858(K4), 1998.

[MAK 62] MAKITA Y., « Localisation directionnelle du son dans un champ sonore stéréophonique », *Revue de l'UER*, juin 1962.

[MAL 92] MALHAM D., « Experience with large Area 3D Ambisonic Sound Systems », *Proceeding of the Institute of Acoustics*, 14-5, p. 209-215, 1992.

[MEH 77] MEHRGARDT S., MELLERT V., « Transformation characteristics of the external human ear », *J. Acoust. Soc. Am.,* 61(6), p. 1567-1576, 1977.

[MOL 92] MOLLER H., « Fundamentals of binaural technology », *Applied Acoustics*, vol. 36, p. 171-218, 1992.

[MOO 01a] MOORER J.A., Extension of the methodod spatial harmonics to three dimensions, non publié, www.jamminpower.com, 2000-2001.

[MOO 01b] MOORER J.A., Ultra-Directional Microphones: Part 4, Sonic Solutions, non publié, www.jamminpower.com, 2000-2001.

[NIC 98] NICOL R., EMERIT M., « Reproducing 3d-sound for videoconferencing: a comparison between holophony and ambisonic », *DAFX98*, novembre 1998.

[NIC 99] NICOL R., EMERIT M., « 3d-sound reproduction over an extensive listening area: A hybrid method derived from holophony and ambisonic », *AES 16th International Conference*, avril 1999.

[POL 00] POLETTI M.A., « A Unified Theory of Horizontal Holographic Sound Systems », *JAES*, vol. 48, n° 12, décembre 2000.

[PUL 97] PULKKI V., « Virtual Sound Source Positioning Using Vector Base Amplitude Panning », *Journal of the Audio Engineering Society*, vol. 45, n° 6, 1997.

[THE 01] THEILE G., « Multichannel Natural Music Recording Based on Psychoacoustic Principles », *IRT*, mai 2001.

[VRI 99] DE VRIES D., BOONE M.M., « Wave Field Synthesis and Analysis Using Array Technology », *Proc. 1999 IEEE Workshop on ASPAA*, octobre 1999.

[WUT 01] WUTTKE J., « General Considerations on Audio Multichannel Recording », *19th AES International Conference*, 2001.

[WIL 00] WILLIAMS M., LE DU G., « Multichannel Microphone Array Design », *108th AES Convention*, 5157, février 2000.

[WIL 01] WILLIAMS M., LE DU G., « The Quick Reference Guide to Multichannel Microphone Arrays – Part I: using Cardioid Microphones », *110th AES Convention*, 5336, mai 2001.

[WK92] WIGHTMAN F., KISTLER D., « The dominant role of low-frequency interaural time differences in sound localization », *J. Acoust. Soc. Am.*, vol. 91, 1992.

[WEN 91] WENZEL E., WIGHTMAN F., « Localization with non-individualized virtual acoustics display cues », *Human Factors in computing systems 8th annual conference*, New-Orleans, 1991.

Chapitre 4

Ordonnancement pour les systèmes musicaux temps réel

4.1. Introduction

Les logiciels musicaux, en particulier les logiciels MIDI, sont souvent amenés à traiter et à produire des flots d'événements datés et organisés dans le temps. Malheureusement, l'ordre de production de ces événements n'est bien souvent pas celui souhaité pour leur restitution. C'est pourquoi il est utile de disposer d'un mécanisme permettant à l'application « d'envoyer des événements dans le futur » et se chargeant de les distribuer au bon moment.

Pour comprendre la nécessité d'un tel mécanisme, prenons l'exemple d'une application qui transforme chaque note reçue en une suite de « rebonds » à la manière d'une balle de ping pong. L'application reçoit un flot d'événements MIDI entrants, et pour chaque note reçue, génère la suite de rebonds qui lui correspond sous la forme d'une séquence de notes de même hauteur, dont le rythme va en s'accélérant pendant que l'intensité des notes diminue progressivement afin de simuler le comportement physique de la balle.

Produire ces rebonds directement dans l'ordre de restitution temporel est assez compliqué dès que les rebonds de plusieurs notes reçues s'entremêlent. Il faut garder la trace de tous les rebonds en cours de simulation et savoir en permanence la date du prochain rebond afin de produire la note correspondante le moment venu. Il serait plus

---

Chapitre rédigé par Yann ORLAREY.

simple de précalculer, à la réception de chaque note entrante, la séquence des rebonds qui lui correspond et de déléguer à un mécanisme spécifique la tâche de conserver les événements le temps nécessaire afin de les délivrer à leur date d'échéance pour le compte de l'application.

Le problème ressemble donc à un problème de tri, mais sur un ensemble d'événements qui évolue au fur et à mesure que de nouveaux événements sont placés dans l'ordonnanceur et que des événements arrivés à échéance en sont retirés. En fait, il se rapproche beaucoup du problème de la gestion des événements dans les logiciels de simulation. Du reste, de nombreuses techniques intéressantes ont été proposées dans le cadre des systèmes de simulation pour gérer l'ensemble des événements en attente (*pending event set*) [JON 86, RON 97].

Malheureusement, des contraintes et des besoins différents rendent les algorithmes utilisés en simulation peu applicables au domaine musical temps réel. D'une part, les systèmes de simulation doivent pouvoir déterminer rapidement le prochain événement à traiter alors que les systèmes musicaux peuvent se contenter de déterminer les événements qui arriveront à échéance à la prochaine unité de temps (car ils sont de toute façon appelés à chaque unité de temps). D'autre part, les systèmes musicaux ont des contraintes temps réel qui supposent l'absence de cas très défavorables et qui interdisent le recours à des opérations lourdes de réallocation mémoire ou de réorganisation des données, comme peuvent le faire les systèmes de simulation.

Dans un contexte musical temps réel, il est important que le coût d'ordonnancement d'un événement soit faible et surtout borné, et ce quelle que soit l'avance avec laquelle un événement est placé dans l'ordonnanceur (c'est-à-dire quel que soit son temps de séjour dans l'ordonnanceur) et quel que soit le nombre d'événements déjà en attente. Dans les paragraphes suivants, nous présentons un algorithme qui possède ces propriétés. Le coût d'ordonnancement d'un événement est en $O(1)$, tant en ce qui regarde l'avance avec laquelle il est ordonnancé que le nombre d'événements déjà en attente. Cet algorithme a initialement été développé dans le cadre du projet MIDILISP [BOY 86]. Il est utilisé depuis plusieurs années dans le système multitâche temps réel MidiShare [ORL 89, ORL 90].

4.2. Spécification du problème

Avant de présenter l'algorithme dans le détail, voyons plus précisément comment le problème se définit, quelles sont les opérations de base que l'on peut vouloir effectuer sur l'ordonnanceur, quels sont les critères de coût à retenir pour une utilisation temps réel, ainsi que l'utilisation typique que l'on peut en faire en prenant pour exemple le système MidiShare.

4.2.1. *Définitions*

A un instant donné, l'état d'un ordonnanceur $S = scheduler[E, D]$ est caractérisé par l'ensemble E des événements en attente et la date courante D. Chaque événement e inséré dans l'ordonnanceur est muni d'une *date d'échéance*, que nous noterons $d(e)$, et qui est la date à laquelle on veut qu'il soit retiré de l'ordonnanceur et distribué à son destinataire. Les dates sont représentées par des entiers non signés de 32 bits. Comme nous le verrons, le fait que les dates soient de taille fixe est important dans la définition de l'algorithme.

Nous appellerons *date de placement*, que nous noterons $p(e)$, la date à laquelle un événement est placé dans l'ordonnanceur, c'est-à-dire la date courante D de l'ordonnanceur au moment de l'insertion de l'événement e. L'*avance* avec laquelle un événement est placé dans l'ordonnanceur (également appelée *durée de séjour*) est la différence entre sa *date d'échéance* et sa *date de placement* : $d(e) - p(e)$. Dans le cadre d'un fonctionnement normal, les événements sont placés avec une *avance* positive : $d(e) > p(e)$. Le nombre d'événements en attente dans l'ordonnanceur $scheduler[E, D]$ à un instant donné est $|E|$.

4.2.2. *Opérations sur l'ordonnanceur*

Munis de ces quelques définitions, nous pouvons décrire plus précisément l'état initial de l'ordonnanceur ainsi que les deux opérations principales d'insertion d'un événement et de retrait des événements arrivés à échéance.

4.2.2.1. *Initialisation de l'ordonnanceur*

INITIALISATION$() \rightarrow scheduler[\emptyset, 0]$: à l'initialisation, l'ensemble des événements en attente est bien évidemment vide et la date courante de l'ordonnanceur est 0.

4.2.2.2. *Insertion d'un événement*

INSERTION$(e, scheduler[E, D]) \rightarrow scheduler[E + \{e\}, D]$: l'événement ordonnancé est ajouté à ceux déjà en attente. Pour que l'opération soit correcte, on suppose que l'événement est placé avec une avance positive, c'est-à-dire que $d(e) > D$.

4.2.2.3. *Avance de l'horloge et retrait des événements arrivés à échéance*

HORLOGERETRAIT$(scheduler[E, D]) \rightarrow scheduler[E - R, D + 1]$ and RETURN(R) avec $R = \{e | e \in E, d(e) \leqslant D + 1\}$: la date courante de l'ordonnanceur est avancée d'une unité et tous les événements arrivés à échéance sont retirés de l'ensemble des événements en attente et retournés.

4.2.3. *Coût d'ordonnancement*

Le coût d'ordonnancement d'un événement e, que nous noterons $C_o(e)$, se décompose en trois parties :

– $C_i(e)$, le coût d'insertion dans l'ordonnanceur,

– $C_m(e)$, le coût de maintient dans l'ordonnanceur jusqu'à l'échéance,

– $C_r(e)$, le coût de retrait de l'événement e de l'ordonnanceur.

Ainsi, on a $C_o(e) = C_i(e) + C_m(e) + C_r(e)$.

Dans le cadre d'une application temps réel, l'objectif sera non seulement de minimiser le coût total d'ordonnancement d'un événement, mais également de faire en sorte que ce coût soit indépendant du nombre $|E|$ d'événements en attente dans l'ordonnanceur ainsi que de l'avance $d(e) - p(e)$ avec laquelle l'événement est placé dans l'ordonnanceur. En d'autres termes, pour satisfaire aux contraintes d'une utilisation temps réel, ce coût doit être borné en toutes circonstances, en particulier lorsque la charge du système est élevée.

4.2.4. *Utilisation de l'ordonnanceur*

Le système MidiShare (voir chapitre ??) fournit un exemple typique d'utilisation d'un ordonnanceur dans un contexte multitâche temps réel. Les applications clientes de MidiShare forment un réseau de communication et, de ce point de vue, l'activité d'une application consiste à recevoir et à émettre des événements datés. Lorsqu'une application émet un événement, celui-ci est placé dans l'ordonnanceur jusqu'à sa date d'échéance avant d'être délivré aux applications destinataires.

Le premier aspect à prendre en considération dans une telle implémentation est la possibilité d'accès concurrent à l'ordonnanceur. En effet, plusieurs applications, éventuellement réparties sur des processeurs différents, peuvent avoir besoin des services de l'ordonnanceur au même moment. Plutôt que de protéger l'accès à l'ordonnanceur par des sémaphores ou d'autres techniques bloquantes, ce qui serait préjudiciable en termes de performances, la solution adoptée par MidiShare consiste à utiliser une liste LIFO intermédiaire dans laquelle sont tout d'abord placés les événements avant d'être insérés dans l'ordonnanceur. Les problèmes d'accès concurrents sont donc déportés sur cette liste intermédiaire. Mais une telle liste peut être implémentée de manière très efficace en utilisant des techniques non bloquantes (*lock-free*).

L'insertion proprement dite dans l'ordonnanceur est directement réalisée par la routine d'interruption *timer* qui cadence le système. Celle-ci commence par récupérer, par une opération atomique, la liste des événements à insérer. Ensuite, elle insère un à un chaque événement de cette liste dans l'ordonnanceur. Puis, elle appelle la méthode

HORLOGERETRAIT afin de récupérer les événements arrivés à échéance. Enfin, ces événements sont distribués aux applications destinataires.

L'enchaînement des opérations réalisées par la routine d'interruption *timer* peut être schématisé comme suit :

$$\text{TIMER=RECUPERATION.}(\text{INSERTION})*.\text{HORLOGERETRAIT.DISTRIBUTION}$$

4.3. L'algorithme d'ordonnancement

Il existe bien entendu de nombreuses façons de traiter le problème d'ordonnancement que nous venons de décrire. Une solution naïve consiste tout simplement à représenter l'ensemble E des événements en attente sous la forme d'une liste d'événements ordonnés dans le temps. Techniquement simple, cette solution ne répond pas aux critères d'efficacité nécessaires et implique un coût d'insertion $C_i(e)$ proportionnel au nombre d'événements en attente dans l'ordonnanceur. Elle n'est donc pas applicable dans des situations avec des contraintes temps réel et un grand nombre d'événements à traiter.

Il existe une solution encore plus simple (mais malheureusement inapplicable dans la pratique, pour des questions de place mémoire), extrêmement efficace en temps de calcul et qui consiste tout simplement à utiliser un tableau avec autant d'entrées que de dates possibles (2^{32} dans le cas présent). Chaque entrée contient la liste des événements en attente à cette date.

L'algorithme que nous allons présenter est une élaboration de cette idée mais nécessitant beaucoup moins de mémoire. Avant d'en faire une description formelle, nous allons étudier cet algorithme à travers l'histoire de *Monsieur Nay* et de la société *DO-IT*.

4.3.1. *L'histoire de Monsieur Nay et de la société DO-IT*

La société DO-IT est un peu curieuse. Son activité principale consiste à recevoir par courrier des ordres à effectuer à des dates précises. Récemment, par exemple, elle a reçu un ordre d'un monsieur âgé, qui pense ne plus être de ce monde d'ici quelques temps. Dans sa lettre, cette personne âgée demande à DO-IT de souhaiter, le 17 février 2018, un joyeux anniversaire à sa fille unique qui aura alors 50 ans.

Un des employés de cette société est chargé, tous les matins, de dépouiller le courrier et de classer toutes les demandes. En outre, il fournit aux autres employés la liste des tâches du jour. Il faut bien entendu qu'il ne perde pas trop de temps à classer les demandes qui arrivent et qu'il retrouve rapidement les demandes du jour. Plusieurs personnes se sont succédées à ce poste, sans beaucoup de réussite. Elles utilisaient

toutes un grand bac dans lequel les fiches étaient classées par ordre chronologique. Elles perdaient beaucoup de temps à parcourir le fichier pour classer les ordres nouvellement reçus. Cette mauvaise gestion a duré jusqu'à l'arrivée à ce poste de Monsieur Nay.

Monsieur Nay, homme organisé, a conçu un nouveau système de bacs de rangement. Il utilise 31 bacs pour les 31 jours du mois, 12 bacs pour les 12 mois de l'année et 100 bacs pour les 100 ans d'activité de la société. Le classement se fait très simplement. Si l'ordre est pour le mois en cours, il le met dans le bac du jour correspondant. Si l'ordre est pour l'année en cours, il le met dans le bac du mois correspondant. Sinon, il met l'ordre dans le bac de l'année correspondante.

Une fois son courrier classé, il n'a plus qu'à prendre les ordres qui se trouvent dans le bac du jour et à les donner aux autres employés. Evidemment, le 1 er du mois suivant, les 31 bacs sont tous vides. Il va donc reprendre tous les ordres du mois qui commence et les reclasser. Suivant le même principe, au début de chaque année, les bacs des jours et des mois sont vides. Notre employé doit donc reclasser tous les ordres de l'année qui commence dans les bacs des mois puis reclasser tous les ordres du mois de janvier dans les bacs des jours.

A ce stade, si nous analysons le coût de traitement d'un ordre, nous nous apercevons que celui-ci n'est manipulé au plus que trois fois, quelle que soit l'avance avec laquelle il est donné. Dans le pire des cas, il sera d'abord placé une fois dans le bac des années, puis une fois dans celui des mois, puis enfin dans celui des jours. De plus, ce coût est indépendant du nombre d'ordres déjà enregistrés.

Le coût de traitement d'un ordre est donc très faible. Par contre, il subsiste un problème qui est le volume de travail accumulé à chaque début de mois et surtout à chaque début d'année. Comme monsieur Nay est un homme astucieux, il a trouvé la solution : répartir son travail de reclassement tout au long de l'année. Pour cela, il utilise des bacs supplémentaires : un deuxième jeu de 31 bacs pour le reclassement du mois suivant et un deuxième jeu de 12 bacs pour le reclassement de l'année suivante.

Tous les jours, Monsieur Nay fait un peu de reclassement. Il prend quelques ordres de l'année suivante et les classe dans le deuxième jeu de 12 bacs. Il prend ensuite quelques ordres du mois suivant et les classe dans le deuxième jeu de 31 bacs. Au début du mois suivant, Monsieur Nay termine complètement le reclassement de ce nouveau mois. S'il a bien réparti son travail, il n'a que peu ou pas de reclassement à faire. Ensuite, il échange les deux jeux de 31 bacs. Suivant le même principe, en début d'année, il termine le reclassement de cette nouvelle année, échange les deux jeux de 12 bacs, puis termine le reclassement du mois de janvier et échange les deux jeux de 31 bacs.

L'analyse des coûts est la même que précédemment – un ordre n'est au plus manipulé que trois fois –, mais cette fois, la charge de travail totale est répartie plus uniformément. Bien évidemment, l'échange des bacs est, au niveau informatique, réduit à un simple échange de pointeurs.

4.3.2. *Implémentation de l'algorithme de Nay*

Voyons comment implémenter concrètement l'algorithme de Nay. Nous allons tout d'abord décrire les structures de données employées, puis les principales méthodes – en particulier l'insertion et le reclassement – dans un formalisme proche du C++.

4.3.2.1. *Structures de données*

L'ordonnanceur (`class sorter`) est organisé autour de quatre systèmes de rangements (`class storage`) dans lesquels les événements vont progresser jusqu'à leur date d'échéance. Chaque rangement dispose de deux zones de stockage (`struct zone`) : primaire et secondaire. Ces zones sont des tableaux de 256 entrées dans lesquels les événements sont organisés en préservant l'ordre d'arrivée (`class fifo`). Les événements (`struct event`) comportent, outre une date d'échéance, un lien de chaînage utilisé pour la constitution de listes chaînées.

4.3.2.1.1. Les événements

Pour les besoins de l'implémentation, un événement est une structure de données ayant essentiellement deux champs : un lien de chaînage et une date sur 32 bits non signée.

```
struct event {
    event* link;
    unsigned long date;
    ...
};
```

Le lien de chaînage sera utilisé pour réaliser des listes chaînées d'événements. Le champ date sur 32 bits indique la date d'échéance de l'événement. Nous aurons besoin d'accéder aux octets qui composent une date. Pour une date d, nous noterons PART$(d, 3)$ son octet de poids le plus fort jusqu'à PART$(d, 0)$ son octet de poids le plus faible, de sorte que $d =$ PART$(d, 3)$*$256^3 +$ PART$(d, 2)$*$256^2 +$ PART$(d, 1)$*$256 +$ PART$(d, 0)$.

4.3.2.1.2. L'ordonnanceur

L'ordonnanceur comporte, outre la date courante, quatre systèmes de bacs de rangement pour les événements en attente ainsi qu'un fifo pour les événements ordonnancés en retard (voir figure 4.1). Réunis, ils forment l'ensemble E des événements en attente dans l'ordonnanceur.

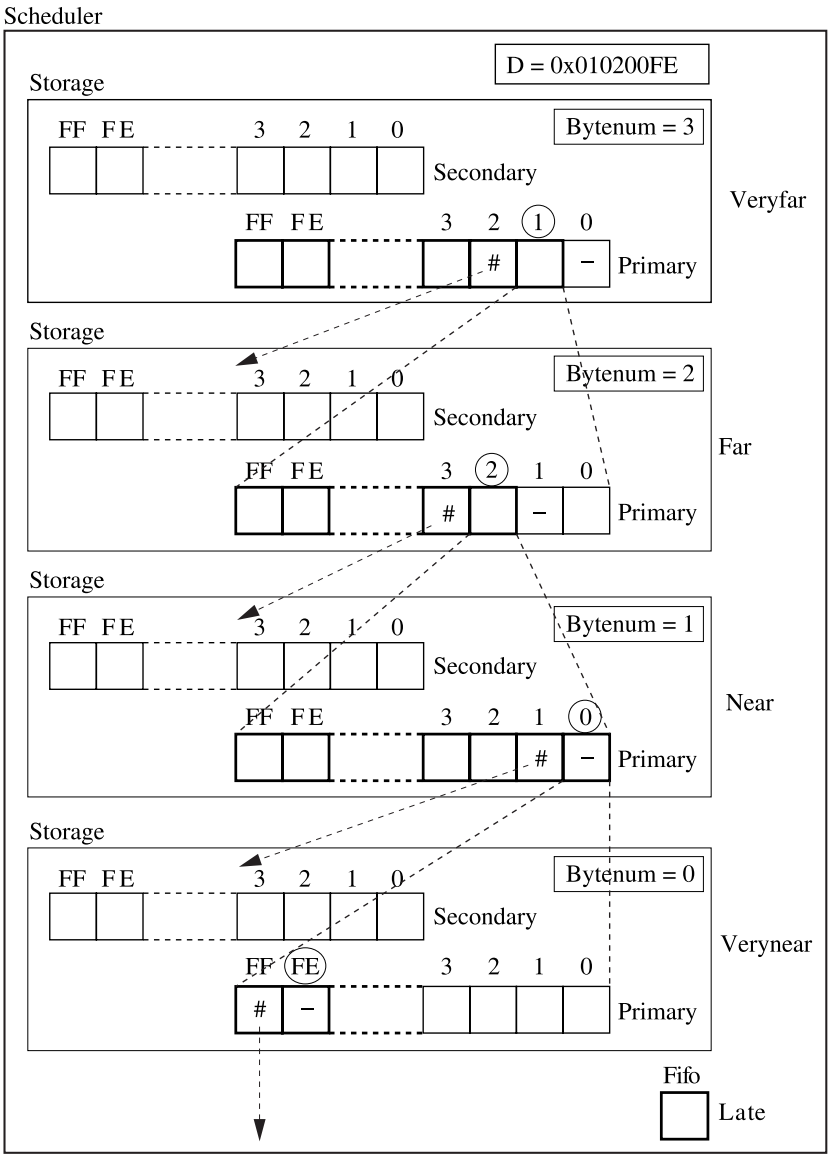


Figure 4.1. *L'ordonnanceur comporte une date courante* `D` *(ici 0x010200FE), quatre structures de rangement des événements :* `veryfar`, `far`, `near` *et* `verynear`, *ainsi qu'un fifo* `late` *pour les événements insérés en retard. Chaque rangement comporte deux zones de stockage :* `primary` *et* `secondary`, *une position temporelle courante* `period` *(matérialisée par un cercle), le fifo d'extraction* `extractable` *(matérialisé par #), ainsi que le numéro d'octet* `bytenum` *à sélectionner dans la date.*

Le principe de l'algorithme étant de maintenir les événements d'autant mieux classés qu'ils sont proches de leur date d'échéance, le rangement `veryfar` correspond aux événements les plus lointains, dont la date diffère de la date courante au niveau de l'octet de poids le plus fort, alors que le rangement `verynear` correspond aux événements presque arrivés à échéance et dont la date ne diffère de la date courante que sur l'octet de poids le plus faible.

```
class sorter {
    unsigned long D;
    storage veryfar;
    storage far;
    storage near;
    storage verynear;
    fifo late;
 public:
    sorter();
    unsigned long date();
    void insert (event* e);
    event* clock_extract();
};
```

Au cours de leur séjour dans l'ordonnanceur, les événements vont successivement passer de `veryfar` à `far`, puis de `far` à `near`, puis de `near` à `verynear`, pour enfin sortir à la date d'échéance.

L'ordonnanceur comporte deux opérations principales. La méthode `insert(e)` insère l'événement `e` dans le rangement et la case appropriés. La méthode `clock_extract()` incrémente la date de l'ordonnanceur, effectue tous les reclassements et retourne la liste des événements arrivés à échéance.

4.3.2.1.3. Les rangements

Comme indiqué au paragraphe précédent, l'ordonnanceur dispose de quatre systèmes de rangement qui contiennent les événements en attente, classés suivant l'éloignement plus ou moins grand de leur date d'échéance. Celle-ci est décomposée en quatre octets numérotés de 0 (octet de poids le plus faible) à 3 (octet de poids le plus fort). Les événements dont la date d'échéance diffère de la date courante sur l'octet 3 sont les plus éloignés et seront rangés dans `veryfar`. Ceux qui diffèrent sur l'octet 2 seront rangés dans `far` et ainsi de suite. Le champ `bytenum` de chaque rangement détermine le numéro de l'octet pris en considération par celui-ci.

On trouve également deux zones de rangement qui vont servir alternativement de zones de rangement primaire et secondaire. La zone secondaire est utilisée pour le travail progressif de reclassement des événements en provenance du

système immédiatement supérieur. Le champ `period` contient l'octet de la date courante déterminé par `bytenum`, de sorte que l'égalité suivante est toujours vérifiée : `D` = `veryfar.period`*256^3 + `far.period`*256^2 + `near.period`*256 + `verynear.period`. Enfin, le champ `extractable` indique le fifo d'où sont extraits les événements.

```
class storage {
    const int bytenum;
    zone    zn[2];
    buffer* primary;
    buffer* secondary;
    fifo*   extractable;
    int     period;
 public:
    storage(int n);
    bool tryput(event* e);
    event* extract(unsigned long cdate, int resort);
    void resort(event* l);
};
```

Les opérations principales sont :

– `tryput(e)` essaye d'insérer un événement dans le rangement. L'opération échoue si la date d'échéance de l'événement est trop proche pour être insérée à ce niveau, auquel cas il faut essayer de l'insérer au niveau inférieur ;

– `extract(date,resort)` extrait tout ou partie des événements extractibles et échange les zones de rangement si nécessaire ;

– `resort(list)` classe une liste d'événements dans la zone secondaire.

4.3.2.1.4. Les zones de rangements

Les zones de rangement sont constituées de 256 boîtes adressées donc sur un octet. Chaque boîte est un fifo afin de maintenir l'ordre de placement des événements.

```
struct zone {
    fifo box[256];
};
```

4.3.2.1.5. Les fifos

L'utilisation de fifos (*first in first out*) plutôt que de simples listes permet de préserver l'ordre d'insertion des événements dans l'ordonnanceur, en particulier lorsque ceux-ci comportent la même date d'échéance.

```
class fifo {
      event* head;
      event* tail;
      void Init();
    public:
      fifo();
      bool   Put(event* e);
      event* GetOne();
      event* GetSome(long n);
      event* GetAll();
      event* GetAllConcat(event* lst);
};
```

Le fifo fonctionne en maintenant deux pointeurs `head` et `tail` sur le premier et le dernier événement d'une liste chaînée. Cinq méthodes principales sont associées à un fifo :

– `Put(e)` permet d'ajouter un événement `e` en queue de fifo ;

– `Get()` permet d'extraire l'événement en tête du fifo ;

– `GetSome(n)` permet d'extraire sous forme de liste chaînée les `n` premiers événements en tête du fifo ;

– `GetAll()` permet d'extraire sous forme de liste chaînée tous les événements du fifo ;

– `GetAllConcat(list)` fonctionne comme `GetAll` mais concatène en plus la liste d'événements passée en argument en queue des événements du fifo (en d'autres termes, `fifo.GetAllConcat(l)` = `fifo.GetAll()++l`).

4.3.2.2. *Les méthodes d'ordonnancement*

4.3.2.2.1. L'insertion d'un événement

La méthode `insert()` insère un nouvel événement dans l'ordonnanceur en fonction de son avance.

```
void sorter insert (event* e)  {
    if (e->date <= D) {
        late.put(e);
    } else {
         veryfar.tryput(e)
        || far.tryput(e)
        || near.tryput(e)
        || verynear.tryput(e);
    }
}
```

Le premier cas traité est celui des événements e en retard ($d(e) \leqslant D$) qui sont placés dans le fifo `late` en attendant la prochaine opération d'extraction. Les événements e en avance ($d(e) > D$) sont, quant à eux, placés dans le rangement approprié en essayant successivement, grâce à la méthode `tryput()`, les rangements *veryfar*, *far*, *near* et *verynear*.

```
bool storage tryput(event* e)  {
    int i = part(e->date, bytenum);
    if (i > period) {
        primary->box[i].put(e);
        return true;
    } else {
        return false;
    }
}
```

Le placement dans `veryfar` réussit si l'octet de poids fort de la date de l'événement est supérieur à celui de la date courante : $\mathrm{part}(d(e), 3) > \mathrm{part}(D, 3)$. Ce même octet sert alors d'indice dans la zone primaire de rangement pour placer l'événement dans le fifo correspondant. Le placement échoue si $\mathrm{part}(d(e), 3) = \mathrm{part}(D, 3)$. Le rangement suivant est alors essayé en utilisant l'octet suivant de la date. A noter que l'une des quatre opérations est assurée de réussir puisque $d(e) > D$ implique qu'il existe $i \in \{3, 2, 1, 0\}$ tel que $\mathrm{part}(d(e), i) > \mathrm{part}(D, i)$.

4.3.2.2.2. L'extraction

La méthode `clock_extract()` incrémente la date de l'ordonnanceur, puis demande à `veryfar` de se mettre à jour et d'extraire des événements à propager dans le rangement suivant grâce à la méthode `extract_update()`. Les événements ainsi extraits sont ensuite distribués dans la zone secondaire de `far` par la méthode `resort()`. La même série d'opérations est accomplie pour les rangements `far`, `near` et `verynear`. Les événements extraits de `verynear` sont ceux arrivés à échéance. Ils sont combinés avec les événements ordonnancés en retard pour donner le résultat final.

```
event* sorter clock_extract()  {
  D = D+1;
  far.resort(veryfar.extract_update(D,RESORT));
  near.resort(far.extract_update(D,RESORT));
  verynear.resort(near.extract_update(D,RESORT));
  return late.getallconcat(verynear.extract_update(D,RESORT));
}
```

La méthode `resort()` reclasse les événements en provenance du niveau supérieur dans la zone secondaire par un adressage direct en fonction de l'octet de la date de l'événement qui lui correspond.

```
void storage resort(event* l)
{
    while (l) {
        event* e = l;
        l = l->link;
        secondary->box[ part(e->date,bytenum) ].put(e);
    }
}
```

La méthode `extract_update()` est plus complexe à cause du mécanisme de reclassement progressif mis en œuvre. La méthode prend deux paramètres : la (nouvelle) date courante de l'ordonnanceur et le nombre d'événements à extraire. La première étape consiste à déterminer s'il s'agit d'un changement de période ou pas. On compare pour cela l'octet concerné de la nouvelle date courante au champ `period`. Si l'on n'est pas arrivé au bout de la période, on se contente d'extraire le nombre souhaité d'événements. Si l'on est arrivé au bout de la période, il faut :

1) extraire tous les événements du fifo extractable afin qu'ils soient totalement reclassés dans le rangement inférieur ;

2) échanger les zones primaires et secondaires si l'on est passé de la période 255 à la période 0 ;

3) mettre à jour le champ extractable pour qu'il désigne le fifo correspondant à la période suivante.

Il est à noter, pour cette dernière opération, que si pour une période p le fifo correspondant à la période suivante est en général `primary->box[`p`+1]`, pour la période 255, il s'agit de `secondary->box[0]`.

```
event* storage extract_update(unsigned long cdate, int resort)
{
    int np = part(cdate,bytenum);
    if (np == period) {
        return extractable->getsome(resort);
    } else {
        event* r = extractable->getall();
        period = np;
        if (np == 255) {
            extractable = &secondary->box[0];
        } else {
```

```
                if (np == 0) {
                    zone* tmp = secondary;
                    secondary = primary;
                    primary = tmp;
                }
                extractable = &primary->box[np+1];
            }
            return r;
        }
    }
```

4.4. Performances

L'analyse du code précédent montre que l'insertion d'un événement représente au plus cinq comparaisons, quelques accès mémoire et un ajout en queue de fifo. Pendant son séjour dans l'ordonnanceur, un événement est, dans le cas le plus défavorable, déplacé trois fois avant de sortir de l'ordonnanceur, soit trois ajouts en queue de fifo. Toutes ces opérations sont indépendantes du nombre d'événements déjà en attente. Le coût maximal d'ordonnancement d'un événement C_o est donc en $O(1)$ quelle que soit la durée de séjour S de l'événement et le nombre N d'événements en attente dans l'ordonnanceur.

4.4.1. *Contexte des mesures*

Pour vérifier expérimentalement ce modèle, nous avons effectué une série de mesures sur un portable Dell C640, disposant d'un Pentium 4 de 1,8 GHz et 256 Mo de RAM, fonctionnant sous Linux Redhat 9 avec un noyau 2.4.20. Les programmes ont été compilés avec GCC 3.2.2 avec l'option d'optimisation -O3. Les mesures ont été réalisées en utilisant la fonction POSIX `clock()` qui renvoie la durée écoulée d'utilisation du processeur par le programme, exprimée en microsecondes. Les mesures portent sur le temps de calcul mis par l'ordonnanceur pour fonctionner pendant T unités de temps, avec une densité moyenne de W événements entrant et sortant à chaque unité de temps et pour une durée moyenne de séjour de $S/2$ unités de temps.

A l'initialisation du test, $N = W * S$ événements sont placés dans l'ordonnanceur avec des dates distribuées aléatoirement entre 1 et $S - 1$ unités de temps. Puis, à chaque unité de temps, et pendant T unités de temps, l'ordonnanceur est appelé afin de récupérer les événements arrivés à échéance qui sont immédiatement réordonnancés de sorte qu'il y ait en permanence N événements dans l'ordonnanceur. Le nombre total d'événements ordonnancés pendant toute la durée du test est $P = W * T$.

A titre de comparaison, nous avons également mesuré les performances d'un autre ordonnanceur basé sur l'algorithme dit du « calendrier » [BRO 88]. Le calendrier est ici un tableau découpé en *jours* et correspondant à une *année* d'activité. Lorsqu'un événement est ordonnancé, il est placé dans la case du jour souhaité indépendamment de son année. Lorsque l'on extrait les événements arrivés à échéance, on prend soin de ne sélectionner que les événements de l'année en cours et de laisser les autres en place. Simple de mise en œuvre, cet algorithme est très efficace lorsque les événements séjournent en général moins d'une année, mais ses performances se dégradent pour des durées supérieures. Pour les tests, nous avons utilisé une année de 2 048 jours afin que les deux algorithmes aient des occupations identiques en mémoire.

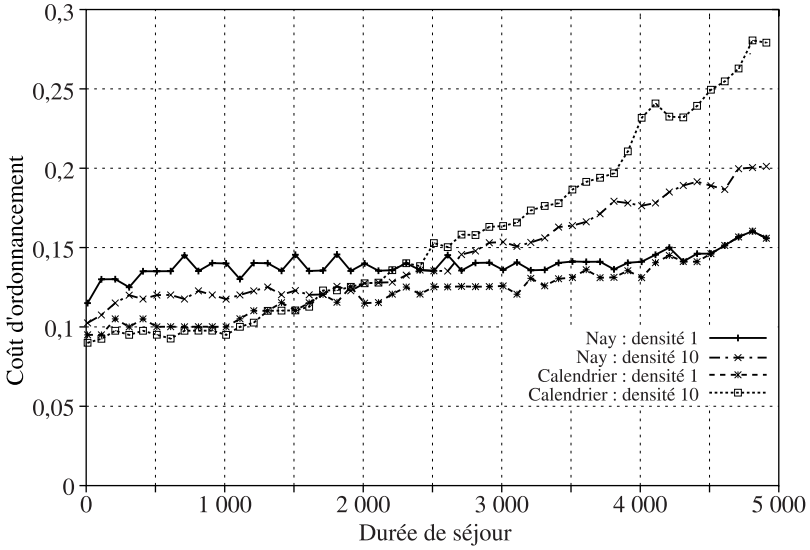


Figure 4.2. *Coût d'ordonnancement d'un événement pour des séjours courts, d'une durée maximale de 0 à 5 000 unités de temps*

4.4.2. *Performances pour des séjours courts*

La figure 4.2 montre l'évolution du coût d'ordonnancement d'un événement (en microsecondes) en fonction du temps de séjour maximal S variant entre 0 et 5 000 unités de temps (soit un temps moyen de séjour variant entre 0 et 2 500 unités de temps). Les mesures ont été effectuées pour une période de fonctionnement $T = 400\,000$. Chaque ordonnanceur est testé pour des densités de 5 et 10 événements par unité de temps. Sur l'ensemble de la période de test, cela représente respectivement 2 000 000

et 4 000 000 événements ordonnancés. Le coût porté en ordonnée est obtenu en divisant la durée du test par le nombre total d'événements ordonnancés. Il est donc légèrement supérieur au coût réel dans la mesure où il intègre une portion du coût fixe liée à l'appel de l'ordonnanceur à vide.

Le graphique montre que l'algorithme du calendrier est légèrement plus efficace pour des séjours qui tiennent dans l'*année* avec $0{,}092\,\mu s$ contre $0{,}107\,\mu s$ pour l'algorithme de Nay.

La tendance s'inverse lorsque la durée maximale de séjours commence à dépasser les 2 500 unités de temps (c'est-à-dire pour des durées moyennes de 1 250 unités de temps).

4.4.3. *Performances pour des séjours moyens*

Sur des séjours plus longs allant jusqu'à 50 000 unités de temps, les mesures montrent des coûts stables conformes au modèle pour l'algorithme de Nay, alors qu'elles montrent, pour l'algorithme du calendrier, des coûts multipliés par 60 par rapport aux cas les plus favorables (voir figure 4.3).

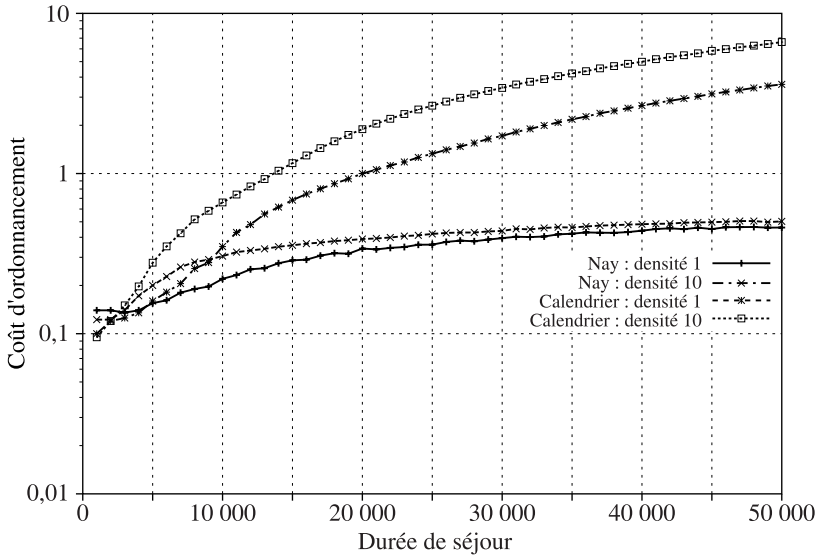


Figure 4.3. *Coût d'ordonnancement d'un événement pour des séjours allant jusqu'à 50 000 unités de temps*

4.4.4. *Performances pour des séjours longs*

Sur de longs séjours, allant jusqu'à 200 000 unités, on mesure la stabilité du coût d'ordonnancement pour l'algorithme de Nay, alors que les coûts pour celui du calendrier sont multipliés par un facteur 100 par rapport aux cas les plus favorables (voir figure 4.4).

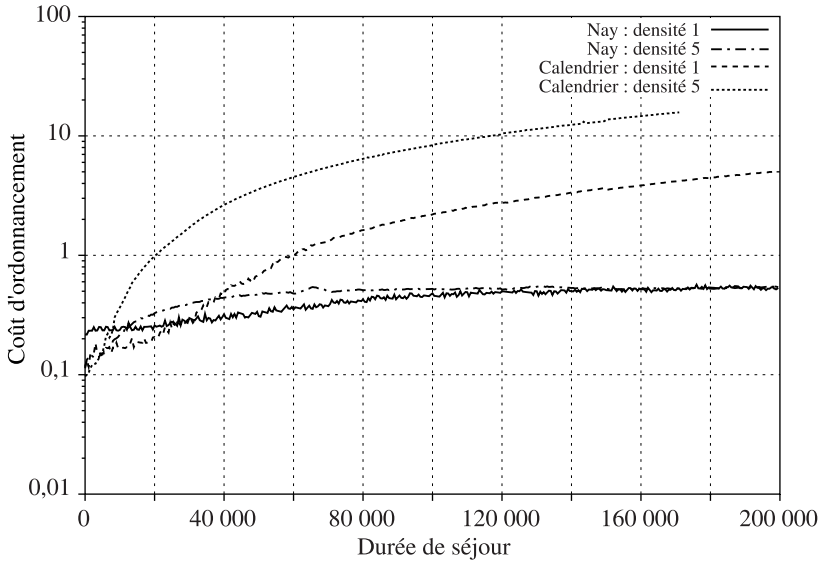


Figure 4.4. *Coût d'ordonnancement d'un événement pour de longs séjours allant jusqu'à 200 000 unités de temps*

4.5. Conclusion

Nous avons présenté un algorithme d'ordonnancement d'événements adapté à une utilisation temps réel grâce à un coût de traitement borné, indépendamment du nombre d'événements en attente et de l'avance avec laquelle les événements sont ordonnancés. Cet algorithme exploite le fait que les dates des événements peuvent être convenablement représentées par des entiers sur 32 bits et que l'ordonnanceur est appelé à chaque unité de temps.

Son principe de fonctionnement est de classer les événements d'autant mieux qu'ils sont proches de leur date d'échéance. Dans la version présentée, les dates des événements sont découpées en quatre octets et un événement est classé au plus quatre fois, par de simples adressages de tableau, avant d'arriver à échéance. D'autres choix

de découpage sont évidemment possibles suivant la nature du problème, les besoins en vitesse et la place mémoire disponible. Comme souvent en informatique, il y a un monnayage entre la vitesse d'exécution et la place mémoire utilisée. Moins il y a de niveaux, plus l'algorithme est efficace mais plus la place mémoire nécessaire est grande. Le cas extrême étant, bien entendu, l'emploi d'un seul niveau.

4.6. Bibliographie

[BOY 86] BOYNTON L., LAVOIE P., ORLAREY Y., RUEDA C., WESSEL D., « MIDI-LISP: A Lisp based music programming environment for the Macintosh », dans *Proceedings of the International Computer Music Conference*, ICMA, 1986.

[BRO 88] BROWN R., « Calendar queues: A fast 0(1) priority queue implementation for the simulation event set problem », *Communications of the ACM*, vol. 31, n° 10, p. 1220-1227, 1988.

[JON 86] JONES D.W., « An empirical comparison of priority-queue and event-set implementations », *Communications of the ACM*, vol. 29, n° 4, p. 300-311, 1986.

[ORL 89] ORLAREY Y., LEQUAY H., « MidiShare: A real time multi-tasks software module for MIDI applications », dans *Proceedings of the International Computer Music Conference*, ICMA, p. 234-237, 1989.

[ORL 90] ORLAREY Y., « An efficient scheduling algorithm for real-time musical systems », dans *Proceedings of the International Computer Music Conference*, ICMA, p. 194-198, 1990.

[RON 97] RÖNNGREN R., AYANI R., « A comparative study of parallel and sequential priority queue algorithms », *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 7, n° 2, p. 157-209, 1997.

Chapitre 5

MidiShare : une architecture logicielle pour la musique

Le développeur d'une application musicale est souvent confronté à des problèmes difficiles à résoudre, notamment parce qu'ils sont relatifs à la maîtrise du temps. Le manque de support des systèmes d'exploitations courants, l'absence de standard, les problèmes de portabilité qui en résultent ne facilitent pas la tâche du programmeur. Nous présentons MidiShare, une architecture logicielle qui a été conçue dans le but de couvrir les besoins des applications musicales de manière homogène, durable et portable. Nous montrons également à travers plusieurs exemples, comment cette architecture facilite le développement, notamment grâce à des mécanismes simples et efficaces de gestion du temps et de communication en temps réel.

5.1. Introduction

Le système d'exploitation d'un ordinateur fournit les services logiciels indispensables à son utilisation. Il organise le partage et la gestion des ressources nécessaires au fonctionnement des applications. Il facilite le travail du programmeur en lui proposant une « machine virtuelle » plus homogène et plus simple à programmer que la « machine réelle » sous-jacente. Il assure une plus grande pérennité aux applications en les isolant autant que possible des particularismes matériels. Ces rôles essentiels du système d'exploitation ne sont malheureusement que très partiellement remplis vis-à-vis des applications musicales.

Chapitre rédigé par Dominique FOBER, Stéphane LETZ, Yann ORLAREY.

Les applications musicales doivent ordonnancer et restituer les événements musicaux avec une grande précision temporelle, de l'ordre de la milliseconde. Elles doivent également travailler en collaboration avec d'autres outils multimédias et satisfaire à différents codes de synchronisation. De ce point de vue, la conception même des systèmes d'exploitation préemptifs multitâches ne permet pas de garantir l'éligibilité d'un processus à une date donnée, ce qui complique singulièrement le développement de processus qui requièrent une grande précision temporelle.

Du fait de contraintes temps réel et d'efficacité, les besoins des applications musicales en termes de mémoire sont également spécifiques et les gestionnaires fournis par les systèmes d'exploitation inadaptés :

– le temps d'allocation d'un bloc mémoire n'est pas déterministe, il peut être long si la mémoire a besoin d'être compactée ;

– pour chaque bloc alloué, il y a un surcoût de plusieurs octets qui rend l'allocation de petits blocs peu rentable, notamment pour les messages MIDI (3 octets de manière générale).

Enfin, les solutions proposées en termes de communication inter applications par les systèmes d'exploitation ne prennent pas en compte la dimension du temps propre aux applications musicales.

La nécessité de recourir à des architectures matérielles et logicielles spécialisées est donc apparue très tôt dans le domaine de l'informatique musicale. Dès la fin des années 1980, apparaissent des architectures ayant pour ambition de répondre à cette nécessité. Ce sont par exemple, *MROS*[1] ou *MIDI Manager* [MM 90], suivis un peu plus tard de *OMS* [OMS 95] ou FreeMIDI[2]. Nous présentons *MidiShare* [ORL 89], un système conçu par Grame en 1989, qui a entre autres la particularité d'être multiplates-formes et qui a évolué jusqu'à aujourd'hui alors que les systèmes cités ci-dessus ont quasiment tous disparus. Nous montrerons ensuite, à travers des exemples simples, comment les problèmes couramment rencontrés dans le cadre du développement d'applications musicales peuvent être aisément résolus grâce au support de cette architecture. Pour finir, nous aborderons quelques points délicats dans la conception d'applications temps réel.

5.2. MidiShare : une architecture logicielle pour la musique.

MidiShare est un système d'exploitation musical, MIDI, multitâches, temps réel et multiplates-formes, qui vient se mettre en complément du système d'exploitation

1. MROS (*MIDI Realtime OS*), développé pour Atari par Steinberg GmbH.
2. *FreeMIDI*, développé par Mark of the Unicorn Inc.

de la machine hôte. Il supporte les principales plates-formes logicielles : GNU/Linux, MacOS et Windows.

5.2.1. *Le modèle conceptuel*

MidiShare est basé sur un modèle client/serveur. Il est composé de six éléments principaux (figure 5.1) : un gestionnaire de mémoire, un gestionnaire du temps associé à un synchroniseur, un gestionnaire des tâches, un gestionnaire de la communication, un gestionnaire de drivers et un ordonnanceur.

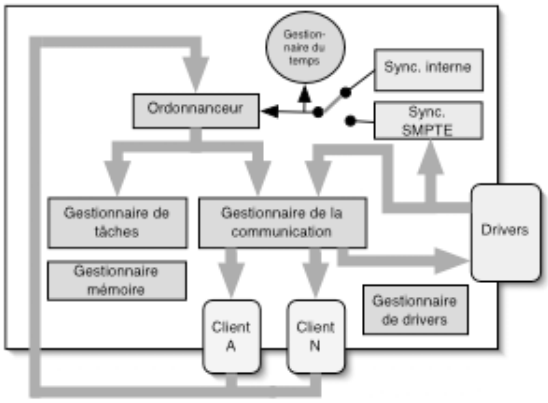


Figure 5.1. *Architecture générale de MidiShare*

5.2.1.1. *Le gestionnaire de mémoire*

La mémoire de MidiShare est basée sur des événements structurés, typés et datés avec une résolution à la milliseconde. Ils recouvrent à la fois les typologies MIDI et MIDI File. Le gestionnaire de mémoire fournit des primitives simples pour l'allocation, la copie et la libération de ces événements, avec un coût d'allocation borné et faible. Il a été conçu pour une utilisation temps réel, il met notamment en œuvre des techniques d'accès concurrent *lock-free* [FOB 02].

5.2.1.2. *L'ordonnanceur*

Il délivre les événements et les tâches à leurs dates d'échéance. Il est possible d'y insérer des événements ou des tâches pour une date quelconque dans le futur. L'algorithme d'ordonnancement [ORL 90] assure un coût de traitement borné et constant par événement, quelque soit la charge de l'ordonnanceur.

5.2.1.3. *Le gestionnaire de la communication*

La communication est basée sur des événements MidiShare, plus simples à manipuler que des paquets d'octets MIDI. Le gestionnaire de communication distribue ces événements, reçus de l'ordonnanceur et des drivers d'entrée, aux applications clientes et aux drivers de sortie, conformément aux connexions courantes.

5.2.1.4. *Le gestionnaire du temps et le synchroniseur*

Le gestionnaire de temps a pour charge de maintenir la date courante du système. Il a une résolution d'une milliseconde. Le synchroniseur supporte le code SMPTE de manière transparente pour les applications clientes.

5.2.1.5. *Le gestionnaire de tâches*

Il a pour responsabilité d'activer pour le compte des applications clientes, les tâches qui sont délivrées à leur date d'échéance par l'ordonnanceur.

5.2.1.6. *Le gestionnaire de drivers*

Les drivers ont un statut particulier puisqu'ils peuvent être intégrés dynamiquement au noyau. C'est le gestionnaire de drivers qui a pour responsabilité de les charger et de les activer.

5.2.2. *Des mécanismes sophistiqués pour la gestion du temps.*

Les applications musicales requièrent une gestion sophistiquée du temps : il ne leurs suffit pas de pouvoir réagir en temps réel aux événements entrants, elles doivent également planifier le futur avec une grande précision. MidiShare fournit plusieurs mécanismes simples et puissants pour la gestion du temps : ce sont les alarmes et les appels de fonction différés dans le temps.

5.2.2.1. *Les alarmes*

5.2.2.1.1. Alarmes de réception

Chaque application cliente peut installer une alarme de réception qui lui permettra de traiter les événements reçus en temps réel. Cette alarme sera activée par MidiShare pour le compte de l'application, au moment où le gestionnaire de communication lui aura distribué un ou plusieurs événements.

5.2.2.1.2. Alarmes de contexte

Chaque application cliente peut installer une alarme de contexte qui lui permettra de réagir aux modifications de son environnement en temps réel. MidiShare active les alarmes de contexte au moment où le changement de contexte s'effectue. Ils

peuvent concerner par exemple la modification des connexions entre applications ou encore l'ouverture d'un nouveau client.

5.2.2.2. *Les appels de fonction différés dans le temps*

5.2.2.2.1. Tâches temps réel

Une application peut effectuer des appels de fonction différés dans le temps en créant une tâche et en confiant son appel à MidiShare pour une date donnée. Lorsque cette tâche arrive à échéance, elle est rendue par l'ordonnanceur et activée par le gestionnaire de tâches pour le compte de l'application qui l'a créée. Cet appel de fonction se fait en temps réel.

5.2.2.2.2. Tâches temps différé

Un mécanisme analogue permet d'effectuer des appels de fonctions en temps différé. A la différence des tâches temps réel, une tâche temps différé, rendue par l'ordonnanceur, n'est pas exécutée par le gestionnaire de tâches mais placée dans la liste des tâches à exécuter de l'application. Son appel est alors de la responsabilité de l'application qui l'a créée. Ce mécanisme permet entre autres d'ordonnancer des tâches qui n'ont pas de contraintes temps réel fortes.

5.2.3. *Une boite à outils canonique*

MidiShare fournit au développeur, un ensemble simple et concis de fonctions qui permettent de satisfaire à l'ensemble des besoins de base des applications musicales. Ces fonctions sont disponibles dans une grande variété de langages (C, C++, Lisp, Java) et de manière uniforme sur les plates-formes logicielles supportées. Les services correspondants peuvent être regroupés dans les catégories suivantes.

5.2.3.1. *Gestion des sessions*

En premier lieu, une application doit s'assurer que MidiShare est installé en mémoire, ce qui est fait par l'appel de la fonction `MidiShare`. Si MidiShare est présent, l'application peut alors ouvrir une ou plusieurs sessions avec la fonction `MidiOpen` qui renvoie un identifiant unique qui servira tout au long de la durée de la session. Toute session ouverte devra être refermée avec la fonction `MidiClose` avant la fin de l'application.

5.2.3.2. *Communication et connexions*

Pour qu'un client MidiShare puisse transmettre et recevoir des événements, il doit tout d'abord être connecté à une ou plusieurs sources et/ou destinations. Un client peut être vu comme une *boite noire*, recevant un flot d'événements en entrée

et produisant un flot d'événements en sortie. Cette *boite noire* peut être librement connectée à d'autres, permettant ainsi de former des réseaux de communication arbitrairement complexes. La communication avec les drivers se fait par le biais d'un *pseudo-client* portant le nom « MidiShare » et ayant le numéro de référence 0. Pour communiquer avec le monde extérieur, un client doit donc être connecté à ce *pseudo-client*.

L'établissement ou la suppression de connexions se fait simplement avec la fonction `MidiConnect` qui prend comme argument les numéros de référence de la source et de la destination, ainsi que l'état désiré de la connexion. La fonction `MidiIsConnected` permet d'interroger l'état de la connexion entre deux clients. Il n'y a pas de restriction sur le nombre de connexions qu'un client peut établir avec d'autres.

Dans certains cas, notamment si l'on veut écrire une application de gestion des connexions, il est important de pouvoir obtenir des informations concernant l'ensemble des clients du système :

– la fonction `MidiCountAppls` permet de connaître le nombre de clients ;

– les fonctions `MidiGetIndAppl` et `MidiGetNamedAppl` renvoient le numéro de référence d'un client en fonction de son numéro d'ordre et de son nom ;

– la fonction `MidiSetName` permet de changer le nom d'un client.

Enfin, un client peut être informé en temps réel des modifications de contexte du système (telles qu'un changement de connexion, l'ouverture ou la fermeture d'un client) : il lui suffit de définir une fonction particulière de gestion des alarmes de contexte, qui pourra être installée avec `MidiSetApplAlarm`. MidiShare se chargera d'activer cette fonction lors de chaque changement de contexte.

5.2.3.3. *Emission et réception*

Une fois connecté, un client peut recevoir et émettre des événements MidiShare. Chaque client possède un FIFO de réception dans lequel MidiShare dépose une copie des événements qui lui sont destinés. Ces événements peuvent provenir d'autres clients ou encore des drivers. La fonction `MidiCountEvs` permet à tout moment de connaître le nombre d'événements en attente dans le FIFO de réception. Ces événements peuvent être retirés du FIFO par des appels successifs à la fonction `MidiGetEv`. Enfin, la fonction `MidiFlushEvs` détruit tous les événements en attente. Les événements retirés du FIFO appartiennent au client correspondant : il peut en disposer librement, mais il est de sa responsabilité de le libérer quand il n'en a plus besoin.

Chaque client peut sélectionner les événements qu'il désire recevoir en utilisant un filtre. Ce filtre agit localement pour son client et n'a donc aucune influence sur les événements reçus par d'autres clients. La gestion de ces filtres se fait avec les fonctions `MidiSetFilter` et `MidiGetFilter`.

Le gestionnaire de temps de MidiShare maintient une horloge interne d'une longueur de 32 bits. Cette horloge est utilisée tant pour dater (en millisecondes) les événements reçus, que pour spécifier la date d'émission d'un événement. Cette horloge peut être interrogée avec la fonction `MidiGetTime`.

Trois fonctions permettent d'émettre des événements :

– `MidiSendIm` : pour la transmission immédiate d'un événement,

– `MidiSend` : pour la transmission d'un événement à sa date courante,

– `MidiSendAt` : pour la transmission d'un événement à une date donnée.

MidiShare se charge automatiquement de délivrer les événements à leur date d'échéance. Grâce à ce mécanisme, un client peut planifier la transmission d'un événement plusieurs jours à l'avance avec une résolution à la milliseconde. Tout événement émis par un client (avec les fonctions `MidiSend`, `MidiSendAt` ou `MidiSendIm`) n'est plus accessible à ce client : il ne doit plus y faire référence sous peine de désorganiser gravement la mémoire MidiShare.

5.2.3.4. *Gestion de la mémoire*

La mémoire MidiShare est organisée en cellules de taille fixe (16 octets) qui permettent de constituer des événements structurés. La fonction `MidiNewEv` permet d'allouer un événement du type désiré. La fonction `MidiFreeEv` permet de le libérer. Un événement alloué peut être dupliqué avec la fonction `MidiCopyEv`. Enfin, l'espace mémoire libre peut être interrogé avec `MidiFreeSpace`, ou étendu avec `MidiGrowSpace`.

Chaque événement est composé d'une cellule d'en-tête qui peut être suivie d'une ou plusieurs cellules d'extension. La figure 5.2 décrit les champs de la cellule d'en-tête :

– le champ `link` est utilisé de manière interne pour chaîner des cellules ;

– le champ `date` contient la date de l'événement ;

– le champ `evType` indique le type de l'événement ;

– le champ `refNum` contient le numéro de référence du client émetteur ;

– le champ `port` indique le port le destination de l'événement ;

– le champ `chan` indique le canal MIDI de destination de l'événement.

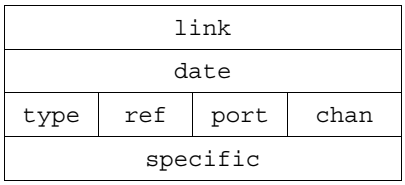


Figure 5.2. *Structure d'une cellule d'en-tête*

Quelque soit le type de l'événement, ces champs sont toujours présents et on peut y accéder directement. Le dernier champ `specific` de la cellule est par contre dépendant du type de l'événement. Dans certains cas, il peut contenir un pointeur sur une ou plusieurs cellules d'extension. Un accès direct à cette partie de la cellule suppose de prendre en compte la structure mémoire correspondant au type de l'événement. C'est pourquoi les fonctions `MidiGetField` et `MidiSetField` permettent de cacher cette structure interne en accédant aux champs spécifiques par indice entre 0 et `MidiCountFields` - 1.

Un certain nombre d'événements ont un nombre de champs variable, c'est le cas par exemple des *System Exclusive*, dans ce cas la fonction `MidiCountFields` renvoie le nombre de champs de la partie variable de l'événement et la fonction `MidiAddField` permet d'ajouter un champ à la fin de la partie variable.

5.2.3.5. *Les séquences MidiShare*

MidiShare permet également la gestion de séquences d'événements ordonnés dans le temps. La fonction `MidiNewSeq` alloue une nouvelle séquence d'événements initialement vide et la fonction `MidiAddSeq` insère un événement dans cette séquence en maintenant l'ordre temporel. La fonction `MidiClearSeq` permet de vider une séquence de son contenu en libérant les événements et la fonction `MidiFreeSeq` libère à la fois la séquence et les événements qu'elle contient.

5.2.3.6. *Les tâches temps réel*

MidiShare fournit à ses clients un moyen simple d'être informé en temps réel de l'occurrence d'un événement par le biais des *alarmes*. Une alarme est une fonction dont l'adresse est passée à MidiShare par un client. Cette fonction sera activée en temps réel par le système lors de l'occurrence d'un événement.

Deux types d'alarmes couvrent les besoins des clients : la première couvre les changements globaux de contexte du système (voir paragraphe 5.2.3.2) et peut être activée avec la fonction `MidiSetApplAlarm`. La seconde est activée avec

`MidiSetRcvAlarm` et informe un client de la présence de nouveaux événements dans son FIFO de réception. Une alarme est toujours activée sous interruption, un client doit donc prendre garde à ne pas faire appel aux ressources du système d'exploitation qui ne sont pas protégées contre la réentrance. A part `MidiOpen` et `MidiClose`, l'ensemble des fonctions de MidiShare sont disponibles dans le contexte des alarmes. Un client peut également accéder à ses variables globales puisque MidiShare restaure le contexte de l'application avant d'activer une alarme.

MidiShare implémente un second mécanisme pour gérer les tâches temps réel : il s'agit des appels de fonctions différés dans le temps. Les fonctions `MidiTask` et `MidiDTask` prennent une date, une fonction et ses paramètres comme arguments, MidiShare crée un événement particulier (de type `typeProcess` ou `typeDProcess`) à partir de ces arguments et passe cet événement à l'ordonnanceur.

A sa date d'échéance, MidiShare active la fonction pour le compte du client correspondant quand il s'agit d'un événement de type `typeProcess`. Pour les événements de type `typeDProcess`, la tâche est mise en attente dans une liste appartenant au client : la fonction `MidiCountDTasks` permet à un client de connaître le nombre de tâches en attente et le cas échéant, la fonction `MidiExec1Dtask` exécute la prochaine tâche en attente.

Dans certaines circonstances, il peut être utile *d'oublier* une tâche programmée mais non encore exécutée grâce à la fonction `MidiForgetTask`. Enfin, la liste des tâches en attente de type `typeDProcess` peut être vidée avec la fonction `MidiFlushDTasks`.

5.3. Exemples

5.3.1. *Un squelette typique*

Une application cliente de MidiShare va typiquement commencer par configurer une session, exécuter ensuite le corps du programme et refermer la session avant de quitter (exemple 5.1). Elle gère des variables globales, dont le numéro de référence du client MidiShare qui servira tout au long de la session.

La configuration d'une session passe par la vérification de la présence de MidiShare, l'ouverture de la session, le positionnement éventuel d'alarmes de réception et/ou d'application, l'installation optionnelle d'un filtre et enfin l'établissement de connexions (exemple 5.2).

```
#include <MidiShare.h>

short myRefNum ; /* numéro de référence du client
MidiShare */
MidiFilterPtr  myFilter=0 ; /* un filtre d'événements
(optionnel) */
MidiName       ClientName = "foo" ; /* le nom du client
*/

int main( int argc, char *argv[])
{
 /* configure une session MidiShare */
 if ( !SetUpMidi() ) return 0 ;
 /*
       corps du programme
 */
 /* referme la session */
 QuitMidi() ;
}
```

Exemple 5.1.

```
Boolean SetUpMidi(void)
{
 /* on vérifie que MidiShare est présent */
 if ( !MidiShare() ) return false ;

 /* on ouvre une session MidiShare */
 myRefNum = MidiOpen (ClientName) ;
 /* on vérifie que l'ouverture est correcte */
 if ( myRefNum < 0 ) return false ;

 /* selon les besoins du client :
       installation d'une alarme de réception */
 MidiSetRcvAlarm (myRefNum, myRcvAlarm) ;
 /*    installation d'une alarme d'application */
 MidiSetApplAlarm (myRefNum, myApplAlarm) ;

 /* le cas échéant, on crée un nouveau filtre */
 myFilter = MidiNewFilter () ;
 if (myFilter) {
       /* on le configure (exemple de configuration ci-
dessous) */
       SetupFilter (myFilter) ;
       /* et on l'installe */
       MidiSetFilter (myRefNum, myFilter) ;
```

```
  }

  /* on établit pour finir les connexions en entrée et en
sortie
     ici avec le pseudo-client MidiShare (numéro de
référence 0) */
  MidiConnect (myRefNum, 0, true) ;
  MidiConnect (0, myRefNum, true) ;
  return true ;
}
```

Exemple 5.2.

La fermeture d'une session consiste à libérer les ressources qui ont pu être allouées dans le cours du programme et à fermer la session MidiShare (exemple 5.3).

```
void QuitMidi()
{
  /* on libère les ressources qui ont été allouées :
     filtres, séquences etc… */
  MidiFreeFilter (myFilter) ;
  /* et on referme la session MidiShare */
  MidiClose (myRefNum) ;
}
```

Exemple 5.3.

Pour former un tout pouvant être compilé et exécuté, les exemples qui suivent présupposent l'utilisation de ce squelette. Les fonctions correspondantes viennent en complément ou en remplacement de fonctions existantes. Afin d'éviter des erreurs de compilation, il sera généralement nécessaire des les réordonner pour que leur déclaration précède leur utilisation. En utilisant un compilateur tel que gcc, vous obtiendrez un fichier exécutable avec la ligne de commande suivante :

```
# l'option -l lie le programme à la librairie MidiShare
> gcc monfichier.c -lMidiShare -o monprogramme
```

5.3.2. *Un écho*

Nous considérons ici que l'écho d'une note est une suite de notes de même hauteur, d'intensité décroissante, plus ou moins espacées dans le temps. Une application, qui génère un écho des notes reçues sur son entrée, a besoin d'installer un

filtre d'événements et une alarme de réception. Il faudra qu'elle dispose également d'une tâche temps réel pour la génération de l'écho. L'application sera paramétrée par deux variables globales : le délai entre deux échos et l'amortissement.

Le filtre de l'application permet de garantir au client MidiShare qu'il ne recevra que des événements de type Note ou KeyOn (exemple 5.4). Un événement de type Note regroupe un KeyOn et un KeyOff, il comprend un champ de durée.

```
void SetupFilter (MidiFilterPtr    filter)
{
 short i ;

 for (i=0 ;i<256 ;i++) {
        /* on rejette tous les type d'événements */
        MidiAcceptType (filter,i,false) ;
        /* on accepte tous les ports */
        MidiAcceptPort (filter,i,true) ;
 }
 for (i=0 ;i<16 ;i++)  /* on accepte tous les canaux MIDI
*/
        MidiAcceptChan (filter,i,true) ;

 /* et on accepte les Notes et les KeyOn */
 MidiAcceptType (filter, typeNote, true) ;
 MidiAcceptType (filter, typeKeyOn, true) ;
}
```

Exemple 5.4.

L'alarme de réception traite les événements reçus en temps réel. Pour chaque événement, elle installe une tâche temps réel qui est en charge de la génération de l'écho. A noter que les événements KeyOn sont transformés en notes auxquelles on affecte une durée fixe.

```
unsigned short delay=100 ;       /* les échos sont
séparés de 100 ms */
unsigned short amortissement=1 ; /* amortissement de la
vélocité */

void myRcvAlarm ( short refnum)
{
 MidiEvPtr ev ;
 while (ev = MidiGetEv (refnum)) {
        /* si l'événement n'est pas une Note c'est un
KeyOn */
```

```
        if ( EvType(ev) != typeNote ) {
              /* il est transformé en Note */
              EvType(ev) = typeNote ;
              /* on lui affecte une durée de 100 */
              Dur(ev) = 100 ;
        }

        /* un KeyOn avec une vélocité de 0 est souvent
utilisé
           à la place d'un KeyOff, d'où cette
vérification */
        if ( Vel(ev) > 0 )
              /* on programme la tâche qui génère l'écho
pour la date de
                 l'événement + le délai courant
                 l'événement à répéter est passé en
paramètre de la tâche*/
              MidiTask (EchoTask, Date(ev) + delay,
refnum, (long)ev, 0,0) ;
        else MidiFreeEv(ev) ;
  }
}
```

Exemple 5.5.

La tâche qui génère l'écho vérifie qu'il reste des échos à jouer, le cas échéant, elle en joue un et se re-programme pour la date de l'écho suivant.

```
void EchoTask (long date, short refnum, long e, long,
long )
{
 /* on récupère l'événement passé en argument */
 MidiEvPtr ev = (MidiEvPtr)e ;
 /* on calcule la nouvelle vélocité de l'événement */
 short v = Vel(ev) - amortissement ;

 /* s'il reste des échos à jouer */
 if ( v > 0 ) {
       /* on met l'événement à jour */
       Vel(ev) = v ;
       /* on émet une copie de l'événement */
       MidiSendIm (refnum, MidiCopyEv(ev)) ;
       /* et on re-programme la tâche  */
       MidiTask (EchoTask, date + delay, refnum, e, 0,
0) ;
```

```
  }
  /* on libère l'événement s'il ne reste plus d'échos à
jouer */
  else MidiFreeEv(ev) ;
}
```

Exemple 5.6.

5.3.3. *Un séquenceur*

Le séquenceur simple, que nous présentons, est une application qui peut enregistrer les événements qui se présentent sur son entrée et qui peut les relire à la demande. L'application va stocker ces événements dans une séquence MidiShare qui doit être allouée à l'initialisation et libérée à la fermeture de la session. Elle gère également un état qui permet de contrôler l'enregistrement ou la lecture. L'enregistrement se fait dans l'alarme de réception (exemple 5.7) et la lecture nécessite une tâche temps réel spécifique.

```
enum { idle, playing, recording } ;

MidiSeqPtr mySeq ;
int state = idle ;
void myRcvAlarm (short refnum)
{
  /* si le séquenceur est en enregistrement */
  if (state == recording) {
        MidiEvPtr ev ;
        /* pour chaque événement reçu */
        while (ev = MidiGetEv(refnum)) {
              /* envoie une copie de l'événement
                 (fonctionnalité de thru optionnelle) */
              MidiSendIm (refnum, MidiCopyEv(ev)) ;
              /* et stocke l'événement dans la séquence
MidiShare */
              MidiAddSeq (mySeq, ev) ;
        }
  }
  /* le séquenceur n'enregistre pas : on libère tous les
événements */
  else MidiFlushEvs (refnum) ;
}
```

Exemple 5.7.

La tâche qui relit une séquence enregistrée est initialisée avec le premier événement de la séquence. Son principe est le suivant : elle joue tous les événements qui sont à la même date. Quand elle rencontre un événement qui n'est plus à la date courante, elle se re-programme pour la date correspondant à cet événement en le passant comme argument du prochain appel (exemple 5.8).

A titre de rappel, les événements sont ordonnés par dates croissantes dans les séquences MidiShare.

```
void PlayTask (long date, short refnum, long e, long a,
long b)
{
 /* on récupère l'événement courant */
 MidiEvPtr ev = (MidiEvPtr)e ;
 /* on préserve sa date */
 long offset, d = Date(ev) ;

 /* on vérifie que le séquenceur joue */
 if (state != playing) return ;

 /* pour chaque événement dans la séquence */
 while (ev) {
        /* on calcule son offset avec le premier
événement joué */
        offset = Date(ev) - d ;
        /* sa date est différente : on sort de la boucle
*/
        if (offset) break ;
        /* on envoie une copie de l'événement */
        MidiSendIm (refnum, MidiCopyEv (ev)) ;
        /* et on passe à l'événement suivant */
        ev = Link(ev) ;                    // go to next
event
 }
 /* s'il reste des événements à jouer, on programme la
tâche
    pour le futur avec l'événement courant comme argument
*/
 if (ev) MidiTask (PlayTask, date + offset, refnum,
(long)ev, 0, 0) ;
 else state = idle ;
}
```

Exemple 5.8.

De manière rudimentaire, le corps du programme peut automatiquement enregistrer au démarrage, puis rejouer ensuite la séquence enregistrée (exemple 5.9) avant de quitter.

```
/* allocation d'une nouvelle séquence */
mySeq = MidiNewSeq () ;
/* changement d'état pour enregistrer */
state = recording ;

printf ("appuyer sur <return> pour arrêter
l'enregistrement") ;
getc(stdin) ;
/* arrêt de l'enregistrement */
state = idle ;

printf ("appuyer sur <return> pour commencer la
lecture") ;
getc(stdin) ;
/* activation de la tâche de lecture (si la séquence
n'est pas vide) */
if (First(mySeq)) {
 state = playing ;
 PlayTask (MidiGetTime(), myRefNum, (long)First(mySeq),
0, 0) ;
}
/* attente de la fin de la lecture */
while (state == playing)
 sleep (1) ;
```

Exemple 5.9.

5.4. Quelques difficultés de la programmation temps réel

Si l'utilisation de MidiShare permet une écriture simple et concise des tâches temps réel, un certain nombre de problèmes restent à résoudre quand on va dans le détail des implémentations. Parmi eux figurent notamment les problèmes de latence, de précision et de dérive temporelle.

5.4.1. *Composer avec la latence*

Nous désignons ici la *latence* comme étant le délai qui sépare la date d'échéance d'une tâche de sa réalisation effective. Ces délais peuvent intervenir à tous les niveaux dans la chaîne de traitement d'un message musical : quand l'ordonnanceur

du système d'exploitation est en jeu (particulièrement dans le cas des langages non temps réel tels que Lisp ou Java) ou encore quand les événements sont délivrés *via* un canal de communication dont les temps de transmission ne sont pas déterministes (par exemple : cas des transmissions sur un réseau). Des techniques particulières doivent alors être utilisées pour compenser la latence et conserver strictement l'ordonnancement temporel des événements qui sont transmis.

Nous allons supposer que nous sommes dans le cas d'un langage non temps réel tel que Java. Nous avons programmé une tâche qui émet un événement à intervalles réguliers. Cette tâche se re-programme elle-même avec la périodicité correspondante. Cependant, à sa date d'échéance, l'activation de la tâche prendra un temps variable, dépendant essentiellement du contexte courant de la machine. Ce temps est généralement borné et nous désignons par *maxlat*, la valeur maximale de ce délai. Pour une périodicité souhaitée *P*, la périodicité réelle de chaque tâche va donc varier en fonction de la latence courante et en conséquence, l'ordonnancement temporel des événements émis sera déformé. Une solution consiste à anticiper la variation de la latence et à décaler les événements dans le futur d'une valeur qui soit au moins égale à *maxlat*, de manière à s'assurer que ces événements ont bien été émis au moment de leur date d'échéance (exemple 5.10).

```
/* la tâche prend comme argument l'événement à émettre,
la période
   la latence maximale attendue */
void myTask (long date, short refnum, long e, long
periode, long maxlat)
{
 /* on envoie une copie de l'événement dans le futur,
décalé de la
    valeur de la latence maximale */
 MidiSendAt (refnum, MidiCopyEv((MidiEvPtr)e), date +
maxlat) ;
 MidiTask (myTask, date + periode, refnum, e, periode,
maxlat) ;
}
```

Exemple 5.10.

Cette technique introduit généralement un délai faible, tolérable dans la plupart des contextes musicaux, ce qui la rend applicable dans bien des cas. Elle est notamment mise en œuvre pour compenser les délais de transmission d'événements musicaux sur Internet [FOB 01].

5.4.2. *Dérive temporelle*

Typiquement, les dérives temporelles affectent les tâches qui se répètent elles-mêmes en faisant usage d'un offset pour calculer la date d'échéance suivante. En effet, la date passée en argument de la tâche (sa date d'échéance) ne correspond pas toujours à la date courante du système. Cela peut provenir des effets de la latence ou encore être le fait d'une tâche qui a été programmée *en retard*.

Reprenons l'exemple de la tâche périodique précédente : si cette tâche faisait usage de la date courante du système pour se reprogrammer, elle dériverait dans le temps en fonction des variations de la latence (exemple 5.11).

La solution consiste ici à faire rigoureusement usage de la date *logique* de la tâche (voir exemple 5.10). La même règle s'applique pour les événements.

```
void myTask (long date, short refnum, long periode, long,
long)
{
 /* la tâche fait usage de la date du système et n'est
donc pas à l'abri de dérives temporelles */
 MidiTask (myTask, MidiGetTime() + periode, refnum,
periode, 0, 0) ;
}
```

Exemple 5.11.

Les approximations de calcul sur les dates constituent également une cause importante de dérive temporelle. Ces approximations interviennent généralement quand il s'agit de convertir une date entre différentes références de temps : du temps musical vers le temps absolu par exemple. Le temps musical fait usage d'unités qui dépendent du tempo courant. Ce tempo peut être exprimé de différente manière : il est représenté en microsecondes par noire dans la norme MIDI File, il consiste en nombre de battues par minute pour les musiciens.

Quelque soit la précision des nombres utilisés pour calculer les dates, les divisions par exemple, produiront généralement des approximations qui cumulées, conduiront à terme à une dérive temporelle. La solution consiste ici à conserver les valeurs ignorées et à les réinjecter dans le calcul quand elles deviennent significatives (exemple 5.12).

```
long long tempo ; /* exprimé en microsecondes par noire
*/
                  /* un tick représente 1/24 de noire */
                  /* le type 'long long' permet d'éviter
*/
                  /* les débordements lors des calculs */
void myTask (long date, short refnum, long ticks, long,
long)
{
 static long reste = 0 ;
 /* conversion en temps absolu (millisecondes) */
 long long offset = (tempo * ticks) / 24000 ;
 /* on garde le reste de la conversion */
 reste += (long)(tempo * ticks) % 24000 ;
 /* et au besoin, on le réinjecte dans l'offset */
 while (reste > 1000) {
       offset += 1 ;
       reste -= 1000 ;
 }
 MidiTask (myTask, date + offset, refnum, ticks, 0, 0) ;
}
```

Exemple 5.12.

5.5. Conclusion

Le développement d'applications musicales requiert une maîtrise du temps qui suppose à la fois une bonne connaissance et une grande pratique des couches les plus basses des systèmes d'exploitation ciblés. Nous avons vu avec MidiShare, que la mise en œuvre d'une architecture logicielle dédiée aux types de problèmes que soulèvent les applications musicales, permet de simplifier le travail du développeur de manière très significative, en fournissant notamment des fonctions de haut niveau

pour la gestion du temps ainsi qu'une représentation structurée des événements musicaux. MidiShare est un logiciel « Open Source », librement accessible à l'adresse suivante : http://www.grame.fr/MidiShare.

Nous avons également vu que l'utilisation de cette architecture ne résout pas tous les problèmes, quelques-uns parmi ceux qui relèvent toujours de la gestion du temps ont été abordés. Ce n'est qu'à travers une bonne connaissance du système, alliée à une expérience assidue que l'on parvient à maîtriser les difficultés de la programmation temps réel. Nous ne saurions trop conseiller aux développeurs de soigner leur code avec la minutie d'un horloger.

5.6. Bibliographie

[FOB 01] FOBER D., LETZ S., ORLAREY Y., « Real Time Musical Events Streaming over Internet », *Proceedings of the International Conference on WEB Delivering of Music, IEEE*, p. 147-154, 2001.

[FOB 02] FOBER D., LETZ S., ORLAREY Y., « Lock-Free Techniques for Concurrent Access to Shared Objects », *Actes des Journées d'Informatique Musicale JIM2002, GMEM,* Marseille, p. 143-150, 2002.

[MM 90] MIDI Management Tools, Apple Computer, Inc., 1990.

[OMS 95] OMS (Open Music System) Programming Interface, Opcode System, Inc, 1995.

[ORL 89] ORLAREY Y., LEQUAY H., « MidiShare: a Real Time multi-tasks software module for Midi applications », *Proceedings of the ICMC, ICMA*, San Francisco, p. 234-237, 1989.

[ORL 90] ORLAREY Y., « An Efficient Scheduling Algorithm for Real-Time Musical Systems », *Proceedings of the ICMC, ICMA*, San Francisco, p. 194-198, 1990.

Chapitre 6

Grammaires, automates et musique

6.1. Introduction

Ce chapitre est un panorama de l'utilisation des grammaires et des automates en informatique musicale, depuis les travaux des pionniers dans les années soixante jusqu'à aujourd'hui. Il comporte un rappel des notions élémentaires de théorie des langages, un bref historique des réalisations qui ont marqué les quatre décennies écoulées, et une présentation détaillée de deux exemples, concernant d'une part l'enrichissement harmonique des grilles de jazz, et d'autre part l'analyse des séquences musicales sérielles. Le code *Common Lisp* ou *Lex* correspondant est disponible sur le *web*, ainsi que des fichiers midi ou audio illustrant ce chapitre (www.info.unicaen.fr/~marc/publi/grammaires). Cette présentation est issue d'un enseignement donné à l'université de Caen pendant une dizaine d'années, dans le cadre de projets de licence et maîtrise d'informatique.

6.2. La hiérarchie de Chomsky

6.2.1. *Grammaires formelles*

On rappelle dans cette partie les principales notions sur les grammaires formelles qui seront utilisées dans la suite. Elles sont tirées des ouvrages classiques [AHO 89, EIL 74, GRO 69].

Chapitre rédigé par Marc CHEMILLIER.

Une grammaire formelle est une description d'un ensemble de séquences de symboles. Les programmes informatiques, par exemple, sont des séquences dont les symboles sont les caractères alphanumériques du clavier, enchaînés selon les règles d'une grammaire formelle. Dans les applications linguistiques des grammaires formelles, les symboles sont les mots du dictionnaire et les classes grammaticales.

Formellement, une *grammaire* est définie par la donnée de deux ensembles disjoints de symboles, les terminaux *T* que l'on note par des minuscules et les non-terminaux *N* notés par des majuscules, d'un ensemble fini de *productions* (ou *règles de réécriture*) de la forme $\alpha \rightarrow \beta$ où α et β sont des séquences de symboles de *N* ou *T*, et d'un symbole particulier *S* non-terminal appelé *axiome*. Si $\alpha \rightarrow \beta$ est une production de la grammaire, la séquence $\gamma\alpha\delta$ peut se dériver en $\gamma\beta\delta$. Cela signifie que le fragment α peut être remplacé par β à l'intérieur des séquences dans lesquelles il apparaît.

Le *langage* engendré par la grammaire est l'ensemble de toutes les séquences de symboles terminaux que l'on peut obtenir par dérivation à partir de l'axiome, en utilisant les productions.

La hiérarchie de Chomsky distingue quatre classes de grammaires formelles, selon la forme de leurs productions [CHO 56] :

– type 0 : aucune restriction,

– type 1 (contextuelles, ou *context-sensitive*) : productions de la forme $\alpha A\beta \rightarrow \alpha\gamma\beta$ où *A* est un non-terminal et γ une séquence avec au moins un symbole,

– type 2 (algébriques, ou *context-free*) : productions de la forme $A \rightarrow \alpha$ où *A* est un non-terminal,

– type 3 (régulières ou linéaires) : productions de la forme $A \rightarrow wB$ ou $A \rightarrow w$, où *A* et *B* sont des non-terminaux, et *w* une séquence de terminaux.

Les grammaires de type 1 sont dites « contextuelles », car les séquences α et β y jouent le rôle de *contextes* dans lesquels on peut remplacer le symbole non-terminal *A* par la séquence γ. Notons que la condition sur le nombre de symboles de γ interdit toute production dans laquelle le membre de droite serait plus court que le membre de gauche.

Les grammaires algébriques (type 2) sont caractérisées par le fait que le membre gauche des productions est réduit à un seul symbole (non-terminal). Les grammaires régulières (type 3) sont des cas particuliers de grammaires de type 2, avec une restriction imposée au membre de droite α, qui ne peut contenir qu'un seul symbole non-terminal *B* situé à la fin.

Si l'on désigne par L_0, L_1, L_2, L_3, les ensembles de langages engendrés respectivement par les grammaires de types 0, 1, 2, 3, on a la suite d'inclusions suivante :

$$L_0 \supset L_1 \supset L_2 \supset L_3$$

Ainsi, tout langage régulier (appartenant à L_3) est aussi un langage algébrique (appartenant à L_2), car les grammaires régulières sont des cas particuliers de grammaires algébriques. Lorsque l'on parle du « type » d'un langage, on désigne le plus petit ensemble qui contient ce langage, dans la suite d'inclusions ci-dessus. Ce type correspond à la grammaire *la plus simple* permettant de décrire ce langage. Les ensembles étant emboîtés les uns dans les autres, un même langage peut être décrit par plusieurs grammaires de types différents, qui ne correspondent pas toutes au type du langage lui-même. Par exemple, les *ensembles finis* de séquences sont des langages de type régulier, comme le langage {*ababb*} réduit à une séquence unique. Pourtant, ce langage peut être engendré par la grammaire $S \rightarrow AB$, $A \rightarrow ab$, $B \rightarrow abb$ qui est algébrique, sa première production contenant deux non-terminaux en partie droite. Mais la grammaire régulière $S \rightarrow abB$, $B \rightarrow abb$ engendre le même langage, qui peut donc être donné par deux grammaires de types différents. Les langages de type algébrique sont caractérisés par des phénomènes de récursion infinie, comme le langage {*ab, aabb, aaabbb,* etc.} engendré par la grammaire $S \rightarrow aSb$, $S \rightarrow ab$. Les langages de type régulier ne comportent pas ce phénomène, et c'est la raison pour laquelle la grammaire algébrique précédente peut être remplacée par une grammaire régulière.

6.2.2. *Equivalence entre grammaires et automates*

La théorie des langages montre qu'il existe une équivalence entre les grammaires et certaines classes d'automates. Les automates sont des modèles abstraits de machines qui lisent des séquences de symboles. L'ensemble des séquences lues par une telle machine est le langage « reconnu » par l'automate. On montre ainsi que les grammaires régulières engendrent les langages reconnus par des *automates finis,* et les grammaires algébriques, ceux reconnus par des *automates à pile*.

Un automate fini est défini formellement par un ensemble fini d'*états*, parmi lesquels on distingue un état *initial* et des états *terminaux*, et par un ensemble de *flèches* étiquetées par des symboles reliant entre eux certains états. Les séquences reconnues par un automate fini sont les successions d'étiquettes obtenues en partant de l'état initial et en suivant un chemin dans l'automate jusqu'à un état terminal.

Voici, figure 6.1, un exemple d'automate à deux états, reconnaissant les séquences qui contiennent au moins un *b*. L'état 1 marqué par une flèche entrante est initial, l'état 2 marqué par un double cercle est terminal.

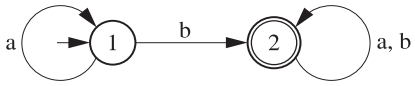


Figure 6.1. *Un automate fini*

Les langages reconnus par automate fini peuvent être représentés par ce que l'on appelle une *expression rationnelle* (théorème de Kleene). Une expression rationnelle est obtenue en appliquant à des ensembles finis de séquences de symboles, les opérations d'union ensembliste (ou de disjonction notée par une barre « | »), de concaténation et d'étoile. L'étoile consiste à itérer un ensemble de séquences, c'est-à-dire à concaténer bout-à-bout un nombre indéfini de séquences de cet ensemble (c'est la formalisation de la notion de « boucle » omniprésente dans les musiques électroniques actuelles). Voici une expression régulière pour l'automate de la figure 6.1 :

a b (a|b)**

Il est parfois utile, dans les applications musicales, de définir un automate avec des étiquettes sur les états plutôt que sur les flèches (voir figure 6.5). Une flèche d'un état à un autre indique que deux symboles peuvent se succéder. On se ramène sans difficulté à la représentation usuelle, en déplaçant les étiquettes des états vers les flèches qui y arrivent, et en ajoutant un état supplémentaire avec des flèches vers tous les états qui ne sont l'arrivée d'aucune flèche.

6.2.3. *Tables de transition et chaînes de Markov*

Pour décrire un automate fini, on utilise parfois une *matrice* ou *table de transition*, dont les lignes et les colonnes sont numérotées par les états de l'automate, et dans laquelle on indique aux croisements les symboles qui permettent de passer d'un état à un autre. Le tableau 6.1 est la table de transition de l'automate en figure 6.1.

	1	2
1	*a*	*b*
2	0	*a, b*

Tableau 6.1. *Table de transition de l'automate*

Lorsque la table de transition est munie de probabilités affectées aux différentes transitions partant de chaque état, on dit que l'automate est une *chaîne de Markov*.

Les tables de transition sont un moyen simple de réaliser informatiquement un automate fini. La fonction Common Lisp ci-après réalise l'automate précédent, par une simple récursion sur la séquence analysée représentée par une liste :

```
(defun automate (etat seq)
  (if (null seq)
      (if (member etat *etats-terminaux*)
          'Reconnu 'Non-reconnu)
      (automate (transition etat (car seq))
                (cdr seq))))
```

La fonction de transition détermine le nouvel état obtenu après lecture d'une lettre, par consultation de la table :

```
(defun transition (etat lettre)
  (cadr (assoc (list etat lettre) *table-transition*
               :test #'equal)))

(setq *table-transition*
'(((1 a) 1)
  ((1 b) 2)
  ((2 a) 2)
  ((2 b) 2)))

(setq *etats-terminaux* '(2))
```

Voici deux exemples d'exécution en partant de l'état initial 1. La première séquence n'est pas reconnue, car elle ne contient pas de *b*. La deuxième s'arrête dans l'état 2 qui est terminal, donc elle est reconnue par l'automate :

```
 ? (automate 1 '(a a a))
Non-reconnu

 ? (automate 1 '(a b a a a))
Reconnu
```

Une variante de la fonction précédente consiste à remplacer la récursion simple associée à une table, par des récursions croisées entre plusieurs fonctions correspondant à chaque état.

La représentation par table pose des problèmes de stockage quand la table est trop grande. Dans ce cas, il faut modifier la fonction de transition ci-dessus, en ne calculant les transitions qu'au fur et à mesure de la lecture, de façon « dynamique ».

6.2.4. *Transductions rationnelles*

Une *transduction rationnelle* est un automate fini « avec sorties », c'est-à-dire dont les flèches sont étiquetées par des couples. Lorsque l'on parcourt un chemin dans une transduction, la succession des éléments en première position dans chaque couple donne la séquence d'entrée, et celle des éléments en deuxième position donne la séquence de sortie correspondante. La figure 6.2 donne un exemple de transduction à trois états. L'étiquette 1 indique qu'il n'y a pas de symbole en sortie.

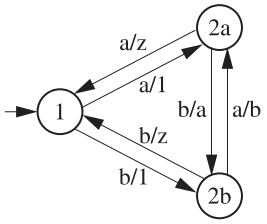


Figure 6.2. *Une transduction rationnelle*

Cette transduction transforme la séquence d'entrée en remplaçant les couples de lettres consécutives identiques par la lettre *z*. Le mécanisme de lecture et d'écriture est le suivant :

– si l'on lit deux lettres consécutives identiques, on écrit la lettre *z* en sortie, et on poursuit la lecture deux lettres plus loin ;

– sinon, on recopie la première lettre en sortie, et on poursuit la lecture en se décalant d'une lettre dans la séquence d'entrée.

Pour la séquence d'entrée *aabaaab,* la séquence de sortie est *zbzab*. Les états 2*a* et 2*b* se « souviennent » de la dernière lettre lue, respectivement *a* ou *b*. Tous les états sont terminaux, avec la convention supplémentaire que si l'on s'arrête dans les états 2*a* ou 2*b,* on rajoute la lettre correspondante à la fin de la séquence de sortie.

Les transductions rationnelles modélisent un type simple de transformation de séquences, dans lequel les symboles de la séquence d'arrivée dépendent de k symboles consécutifs dans la séquence de départ (elles généralisent la notion de *mapping* associant un à un les symboles d'entrée et de sortie, ce qui correspond à $k = 1$). On montre en théorie des langages qu'une transduction rationnelle conserve

le type des langages, c'est-à-dire qu'elle transforme tout langage régulier ou algébrique en un langage de même type. Aussi est-il parfois utile de remplacer l'automate d'un langage par celui d'un autre langage choisi pour être plus facilement implémentable, en combinant celui-ci à une transduction rationnelle qui passe de l'un à l'autre.

Dans les méthodes utilisées pour la génération de séquences musicales, il est fréquent que le problème soit ainsi « factorisé » en plusieurs transductions rationnelles, qui effectuent des traitements successifs. Le plus souvent, la génération d'un langage musical est obtenue par une grammaire ou un automate produisant un « squelette » de ce langage (par exemple des profils mélodiques ou des successions de degrés harmoniques), puis une ou plusieurs transductions permettent d'« habiller » les séquences obtenues pour fabriquer de véritables séquences musicales. Cette construction est schématisée par la figure 6.3.

Figure 6.3. *Factorisation d'un générateur musical en traitements successifs*

On verra que les exemples de Barbaud, de Cope, et de la génération de grilles de blues présentés ci-après illustrent cette construction fondamentale.

6.2.5. *Notion de* parsing

Les grammaires et les automates peuvent être utilisés aussi bien dans le sens de la génération que de l'analyse. Dans ce dernier cas, un « analyseur de séquences » (ou *parser* en anglais) a pour tâche de déterminer si une séquence peut être obtenue par dérivation à partir de l'axiome et des productions d'une grammaire. Les automates finis ou à pile sont des modèles d'analyseurs efficaces pour les grammaires de types 3 et 2. En revanche, pour les grammaires plus complexes (types 1 ou 0), il n'existe pas de méthode efficace pour construire un tel analyseur.

La possibilité de réaliser des analyseurs efficaces explique que les grammaires de types 2 et 3 soient utilisées dans la compilation des programmes informatiques. Un compilateur fonctionne en deux phases. Tout d'abord, le programme à compiler est analysé par un automate fini (plus précisément une transduction rationnelle), qui reconnaît les unités lexicales du langage de programmation (mots-clés, opérateurs, identificateurs, constantes, etc.) et élimine les commentaires, sauts de lignes, etc. La séquence d'unités ainsi construite est ensuite traitée par une grammaire algébrique, qui effectue l'analyse syntaxique, c'est-à-dire le contrôle des structures imbriquées

(début-fin, si-alors, parenthésages, etc.). Les commandes Lex et Yacc du système UNIX permettent de construire des analyseurs pour chacune de ces deux phases. Lex construit un automate fini à partir d'une description sous forme d'expression régulière. Yacc construit un automate à pile à partir d'une grammaire algébrique BNF (Backus-Nauer-Form).

Le listing ci-après est une description de l'automate ci-dessus dans le format utilisé par Lex, qui comporte trois parties séparées par des % % : les déclarations de variables auxiliaires sous forme d'expressions régulières, puis les expressions régulières reconnues par l'automate associées à des actions à exécuter sous forme de code C, et enfin les procédures complémentaires en C. La première partie, facultative, est absente dans l'exemple ci-dessous :

```
 % %
a*b(a | b)*    { printf(« Reconnu\n ») ; return 1 ; }
\n                    { printf(« Non reconnu\n ») ; return
1 ; }
.               ;

 % %
int yywrap()   { return 1 ;}
main()         { printf(« >« ) ; yylex() ; }
```

Ce fichier est donné en entrée à la commande Lex, qui produit en sortie un programme C, donnant lui-même après compilation, une commande exécutable qui réalise l'automate décrit dans le fichier. L'exécution de la commande flex -t toto.l > toto.c ; cc toto.c -o toto fabrique une fonction toto réalisant l'automate dont la description est contenue dans le fichier toto.l. Les deux exemples suivants reprennent ceux de la fonction Common Lisp discutés plus haut :

```
$ toto
> aaa
Non reconnu

$ toto
> abaaa
Reconnu
```

6.3. Grammaires formelles et musique

6.3.1. *L'article de Curtis Roads*

Dans cette partie, on retrace quelques étapes historiques de l'utilisation musicale des grammaires et des automates, en s'arrêtant sur deux automates particuliers

(Barbaud, Cope). Il s'agit d'un choix personnel de quelques exemples significatifs parmi beaucoup d'autres possibles. Les paragraphes suivants indiquent également des ouvrages généraux dans lesquels le lecteur trouvera des références complémentaires.

Les grammaires musicales ont fait l'objet d'un article de synthèse à la fin des années soixante-dix, dans lequel Curtis Roads dressait un bilan des recherches menées durant les deux décennies précédentes [ROA 79]. Il rappelait les principales notions concernant les grammaires formelles, et envisageait différentes manières de les appliquer à la musique en comparant leurs avantages respectifs. Il insistait sur le fait que pour construire une grammaire musicale, il n'y a pas une manière unique de choisir les *symboles*, ceux-ci pouvant aussi bien être des notes que des objets plus complexes comme des chiffrages d'accords, des sections d'une forme musicale, etc., selon l'utilisation que l'on veut faire de cette grammaire. Les principales références présentées dans l'article sont celles de Ruwet [RUW 71], Nattiez [NAT 75], Laske [LAS 73], Smoliar, Moorer [MOO 72], Winograd [WIN 68], Lerdahl et Jackendoff, et Roads. Ce panorama concluait une période marquée par le rayonnement de la linguistique, particulièrement sensible dans les travaux d'analyse paradigmatique développés en France par Nicolas Ruwet.

Parmi les recherches citées par Roads, celles de Lerdahl et Jackendoff ont donné naissance quelques années plus tard au livre *A Generative Theory of Tonal Music* [LER 83] qui a marqué les études de psychologie de la musique. Mais ce travail ne contribue pas directement aux recherches sur les applications musicales des grammaires formelles, bien qu'il s'inspire de la linguistique, car son propos est plutôt de modéliser de façon non-complètement formelle la perception des structures métriques et tonales de la musique.

Au cours des deux décennies suivantes (années 1980 et 1990), divers ouvrages récapitulatifs ont vu le jour [BAR 84, CROSS 91, SCH 93], regroupant des sélections d'articles qui traitent de ce sujet. Ces recherches concernent un large éventail de musiques, incluant le jazz, comme on le verra dans la partie suivante, et les musiques non occidentales [BEC 79, BEL 92, HUG 91, EST], et montrent la persistance de l'intérêt porté aux grammaires formelles, dans un contexte où d'autres modèles concurrents ont vu le jour, notamment la programmation logique avec contraintes.

Parallèlement aux travaux se référant explicitement aux grammaires, une autre tradition de recherches a développé le point de vue des automates, formellement équivalent comme il a été rappelé plus haut. Xenakis a été un pionnier dans l'utilisation musicale des automates avec probabilités de transition, c'est-à-dire des chaînes de Markov [XEN 63], dont il s'est servi pour calculer des successions de nuages de sons dans *Analogiques A* (1958), ou de glissandi enchevêtrés dans *Syrmos* (1958). Barbaud a également utilisé les chaînes de Markov dès les années soixante,

pour construire des générateurs de musique tonale [BAR 65, BAR 68]. Les automates finis, quant à eux, interviennent dans un joli travail de Greussay sur la modélisation d'une pièce des *Mikrokosmos* de Bartok [GRE 73, RIO 79].

Aujourd'hui, les travaux de Cope sur la simulation stylistique adoptent des modèles apparemment plus complexes, comme les *réseaux de transition augmentés* [COP 91], qui se ramènent dans certains cas à des automates finis au sens usuel, comme on en verra un exemple au paragraphe 6.3.2.2.

6.3.2. *Deux automates musicaux*

6.3.2.1. *Barbaud*

Cette section présente de façon plus détaillée la construction de deux automates musicaux. Le premier fait partie d'un ensemble d'automates de musique tonale décrits par le compositeur Pierre Barbaud (mort en 1990). Comme tous les automates de Barbaud [BAR 65, BAR 68, BAR 75, BAR 93], celui-ci est « factorisé » en plusieurs traitements successifs, selon le principe décrit figure 6.3. Tout d'abord, une chaîne de Markov produit une séquence d'accords (une sorte de « basse chiffrée ») à partir d'une table d'enchaînements munie de probabilités, dans laquelle on effectue des tirages aléatoires successifs. Cette séquence est ensuite traitée en complétant les accords et en ajoutant des motifs mélodiques prédéfinis, par tirages dans des tables de positions d'accords et d'ornements.

La figure 6.4 est le début d'un contrepoint original de Barbaud à trois voix. Il fait partie d'un ensemble de fichiers calculés par lui, contenant des séquences musicales codées au format de la machine de synthèse *Biniou* [BRO 98]. Le programme qui a produit ce contrepoint à trois voix est partiellement décrit dans le chapitre 5 du *Vademecum de l'ingénieur en musique* [BAR 93]. La chaîne de Markov engendre des successions d'accords parfaits en rondes, tels que ceux des mesures un, trois ou quatre dans l'exemple ci-dessus. La table de transition comporte 160 couples définissant les enchaînements d'accords possibles, et il n'y a pas d'états terminaux (l'exemple de la figure 6.4 s'interrompt brusquement sans cadence conclusive à la quatre-vingtième mesure).

L'ajout d'ornements fonctionne en traitant la séquence harmonique par blocs de deux accords consécutifs. Elle tire des motifs mélodiques dans la table pour monnayer les trois notes de ces deux accords (ajout de retards, broderies, notes de passage), en fonction de leurs chiffrages et de l'intervalle entre leurs notes de basse. Il se peut, selon le tirage effectué, que l'un des deux accords ne soit pas modifié par le traitement, auquel cas il reste en rondes. Pour économiser l'espace utilisé par la table, les motifs sont stockés à une transposition près, c'est-à-dire qu'au cours de la génération, ils sont transposés de manière à s'ajuster aux accords de la séquence.

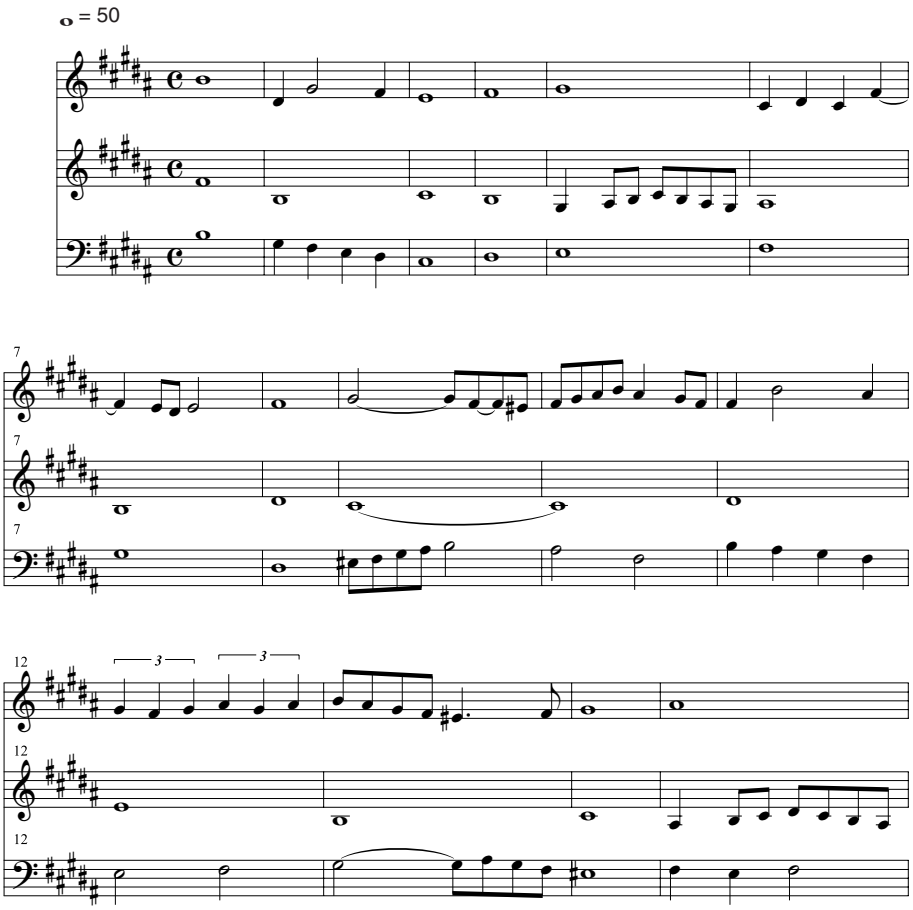


Figure 6.4. *Extrait (mesures 1 à 15) du contrepoint LCONT1.B3I de Pierre Barbaud*

Comme dans la plupart des programmes de Barbaud, le seul paramètre fixé par l'utilisateur est le nombre de mesures de la séquence (quatre-vingt dans l'exemple de la figure 6.4). Les tables utilisées par ce programme ne sont malheureusement pas publiées dans le livre de Barbaud, qui insiste sur l'idée que les tables données par lui ne servent qu'à illustrer une *méthode,* et que le lecteur est invité à imaginer ses propres tables d'enchaînements d'accords et de motifs mélodiques.

6.3.2.2. *Cope*

L'automate de Cope présenté ici fait partie d'un générateur d'inventions à deux voix, dont le code complet en Common Lisp est publié dans le chapitre 4 de

[COP 91]. Le programme recombine des motifs tirés d'un dictionnaire de figures mélodiques empruntées aux inventions de Bach, que Cope appelle des « signatures ». Comme le programme de Barbaud, le générateur est « factorisé » en plusieurs composantes (selon le principe décrit figure 6.3) : un automate fini utilisé dans le sens de la génération produit des séquences qui sont habillées mélodiquement au cours d'un traitement ultérieur, puis transformés par un module de mise en forme contrapunctique. Le programme comporte également un module de *pattern-matching* indépendant, qui détermine les motifs communs à plusieurs inventions de Bach.

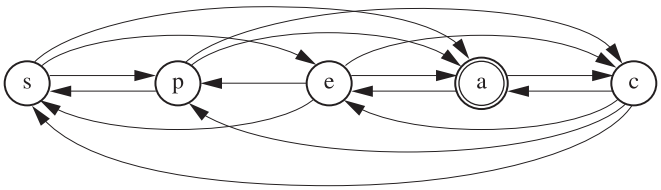


Figure 6.5. *Automate SPEAC de David Cope*

Dans le programme de Cope, le rôle de l'automate fini est limité à la génération de « profils ». Il produit des suites de lettres désignant des intervalles exprimés en demi-tons : $s = 0$, $p = 2$ ou -2, $e = 1$ ou -1, $a = 4$ ou -4, $c = 3$ ou -3. Cet automate est représenté figure 6.5. Tous les états sont initiaux. L'état terminal est *a,* avec la convention que l'on ajoute la lettre *c* après le dernier *a* produit. Le fonctionnement de l'automate est un parcours aléatoire jusqu'à ce que le nombre de *a* obtenus atteigne une certaine valeur. Il peut arriver qu'il tourne indéfiniment si les tirages ne donnent pas suffisamment de *a*.

Le traitement effectué ultérieurement « habille » le profil obtenu par un simple remplacement des lettres une par une. Pour chaque lettre, on cherche dans le dictionnaire un motif mélodique dont la distance entre les notes extrêmes soit égale à l'intervalle correspondant. S'il n'y en a pas, on met à la place un motif tiré au hasard. Le résultat produit à l'issue de cette phase est une simple ligne mélodique.

Le module de traitement contrapuntique prend cette ligne en entrée et la place dans la voix supérieure de l'invention, ainsi que dans la voix inférieure après transposition à l'octave inférieur et décalage d'une mesure. Dans la voix supérieure, les mesures paires sont remplacées par des noires à un intervalle de dixième au-dessus des notes de la voix inférieure. Dans la voix inférieure, les mesures impaires (exceptée la première) sont remplacées par des noires une sixte au-dessous des notes de la voix supérieure. Les deux voix sont tronquées par une cadence conclusive, insérée après un nombre de temps fixé par l'utilisateur.

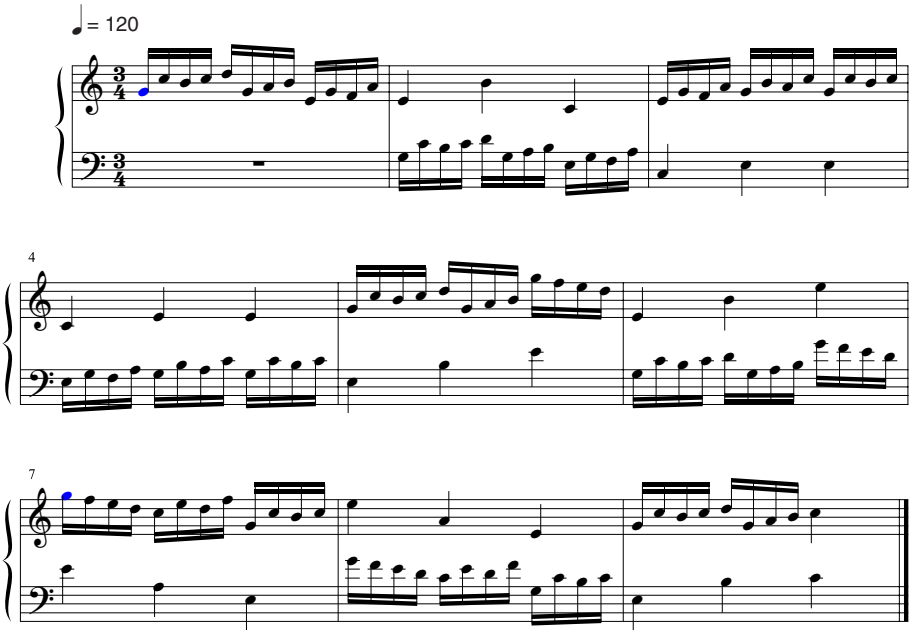


Figure 6.6. *Un exemple d'invention à deux voix (Cope)*

La figure 6.6 montre une séquence produite par le programme de Cope. Ici, le dictionnaire contient des motifs de huit notes consécutives tirés de l'*Invention n° 1* de Bach. Le nombre de *a* fixé pour le profil est dix, et le nombre de temps après lesquels l'invention est tronquée est vingt. Ces trois paramètres (nombre de notes des motifs, nombre de *a* du profil, nombre de temps de l'invention) sont déterminés par l'utilisateur du programme.

L'automate joue un rôle secondaire dans la construction de l'invention, l'essentiel du travail étant réalisé par le module de traitement contrapuntique. Voici le profil engendré par l'automate pour l'exemple de la figure 6.6, avec les intervalles des motifs associés aux lettres. C'est une alternance de *a* et *c*, mais les lettres entre parenthèses correspondent à des motifs qui disparaissent entièrement de la voix supérieure lors du traitement des mesures paires. Les crochets indiquent la portion de la séquence tronquée par la cadence conclusive. La lettre *a* est toujours associée au même motif (dont l'intervalle est 4). En revanche, la lettre *c* est associée à plusieurs motifs (intervalles 8, 5, –2) qui ne correspondent jamais à l'intervalles requis pour cette lettre (3 ou –3). Aussi, le « profil » produit par l'automate ne conditionne-t-il que partiellement le contour mélodique final :

a c (*a*) *c a* (*c*) *a c* (*a*) *c a* (*c*) *a* [*c a c a c a c*]

4 8 (4) 8 4 (5) 4 – 2 (4) – 2 4 (8) 4 [8 4 – 2 4 8 4 8]

Cope précise que le programme présenté ici est incomplet, car il manque des procédures compositionnelles (marches, plan tonal) nécessaires pour construire des inventions plus élaborées comme les deux exemples publiés dans son livre [COP 91, p. 142 et 148].

6.4. Grammaire de substitutions de jazz

6.4.1. *Règles de Steedman*

Cette partie est consacrée à l'étude d'une grammaire de substitutions de jazz, et à la réalisation d'un générateur et d'un analyseur de grilles, dont le code complet est disponible à l'adresse *web* indiquée en introduction. Les substitutions utilisées par les musiciens de jazz consistent à enrichir les grilles harmoniques sur lesquelles ils improvisent, en remplaçant certains accords par d'autres. Elles se prêtent bien à une description au moyen de règles de réécriture, qui permettent de les implémenter sous forme de grammaires formelles. La grammaire étudiée ici est due à Steedman, qui a proposé en 1984, une liste de six règles engendrant des variantes harmoniques de la grille standard du blues de douze mesures [STE 84]. Son article a eu un impact important sur d'autres travaux ultérieurs de Johnson-Laird et Pachet [JOH 91, PAC 98a, PAC 98b], et il a été complété par Steedman lui-même [STE 96]. La description du générateur de substitutions présentée ici est publiée dans [CHE 03].

La grammaire de Steedman est obtenue à partir d'un corpus de neuf grilles de blues middle-jazz et bop empruntées à un ouvrage de Coker sur l'improvisation jazz [COK 64]. Steedman montre que l'on peut dériver toutes les grilles répertoriées par Coker à partir d'une unique grille de blues rudimentaire de trois accords, en utilisant un ensemble de six règles de réécriture. Ces six règles ne jouent pas des rôles de même importance. Deux d'entre elles sont spécifiques : l'avant-dernière ne concerne qu'une grille du corpus et apparaît d'une certaine manière *ad hoc*, et la dernière introduit des accords de septièmes diminuées, ce qui la rend inopérante pour l'analyse de grilles harmoniques qui n'en comportent pas. Dans ce qui suit, nous nous restreindrons aux quatre premières règles publiées par Steedman, et c'est cet ensemble de règles que nous appellerons ici « la grammaire de Steedman ».

Les accords sont notés par des chiffres romains, comme des degrés dans une tonalité indéfinie, auxquels on ajoute éventuellement une abréviation indiquant le type de l'accord choisi parmi trois possibles : majeur (type par défaut), mineur septième (noté m7), et septième de dominante (noté 7). Pour donner une forme plus

concise à la grammaire, Steedman utilise une variable *x* désignant l'un quelconque des degrés, et des fonctions *Dx, Stbx, Sdx* désignant respectivement la dominante, la sus-tonique bémol, et la sous-dominante de *x*, dont le calcul s'effectue facilement sur le cercle présenté en figure 6.7 : *Dx* (respectivement *Stbx, Sdx*) est obtenu à partir de *x* par une rotation de sept unités dans le sens horaire (respectivement une unité et cinq unités), par exemple *Dx* = VII pour *x* = III.

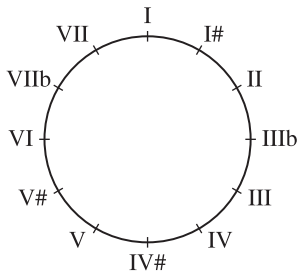


Figure 6.7. *Degrés des accords d'une grille de jazz*

Dans cette grammaire, tous les accords sont considérés comme des symboles non-terminaux. Chaque règle donne naissance à douze règles différentes, lorsque la variable *x* prend comme valeur les douze degrés possibles (et lorsque l'on adapte en conséquence les degrés *Dx, Sdx, Stbx*). Une variable supplémentaire *w* intervient dans la règle 3, pour désigner un accord indéterminé, dont le degré n'a pas été modifié par une règle appliquée précédemment, et dont le chiffrage n'est pas 7. Les règles 1 et 2 ont pour effet de diviser l'unité de longueur en unités plus petites (mesures, demi-mesures). La règle 2 introduit de plus un accord nouveau (sous-dominante), nécessaire pour expliquer le IV apparaissant en deuxième mesure du blues, comme on le voit dans le tableau 6.2, mais il faut noter que son usage réitéré a tendance à modifier profondément le rythme harmonique de la grille. Les règles 3 et 4 sont très utilisées par les musiciens de style bop, et correspondent respectivement à la « préparation II-V » et à la « substitution au triton ».

Règle 1 : $x \rightarrow x\ x$
$x7 \rightarrow x\ x7$
$xm7 \rightarrow x\ xm7$

Règle 2 : $x \rightarrow x\ Sdx$
$x7 \rightarrow x7\ Sdx$
$xm7 \rightarrow xm7\ Sdx$

Règle 3a : $w\ x7 \rightarrow Dx7\ x7$
$w\ x7 \rightarrow Dxm7\ x7$

Règle 3b : $w\ xm7 \rightarrow Dx7\ xm7$

Règle 4 : $Dx7\ x7 \rightarrow Stbx7\ x7$
$Dx7\ x \rightarrow Stbx\ x$
$Dx7\ xm7 \rightarrow Stbxm7\ xm7$

L'axiome *S* de la grammaire est associé à une règle supplémentaire, qui donne le schéma de base du blues (en six unités de deux mesures) :

$S \rightarrow$ I I7 IV I V7 I

La grammaire sous-entend un ensemble de règles implicites par lesquelles chaque accord non-terminal se réécrit comme une copie terminale de lui-même. Les symboles terminaux sont donc des copies des non-terminaux (rappelons que les ensembles de terminaux et de non-terminaux sont supposés disjoints), c'est-à-dire finalement les accords eux-mêmes. Notons que dans le blues, les accords majeurs sont souvent joués de façon « bluesy », c'est-à-dire qu'ils sont identifiés à des accords de septième de dominante. L'application des règles de Steedman, en revanche, distingue ces deux types d'accords, majeur non chiffrés d'un côté, et septième de dominante chiffrés 7 de l'autre. C'est pourquoi Steedman introduit dans son article un chiffrage 7', absent des règles reproduites ici, pour indiquer qu'un accord traité comme majeur dans l'application des règles peut être joué comme septième de dominante en fin de processus. Cet accord chiffré 7' est une sorte de symbole terminal résultant de la réécriture « bluesy » d'un accord majeur non terminal.

Les règles de Steedman sous-entendent également une hypothèse essentielle sur le plan des durées : chaque substitution s'effectue *à durée constante*. Cela signifie que dans les règles ayant un symbole de plus à droite qu'à gauche (règles 1 et 2), les deux accords de droite sont moitié plus courts que celui de gauche. Ces deux règles permettent ainsi de diviser les unités produites par l'axiome (deux mesures) en mesures simples d'abord, puis en demi-mesures, etc. Il est rare que la grille soit divisée au-delà de la demi-mesure, donc l'application potentiellement infinie du processus de division récursif défini par les règles 1 et 2 ne fonctionne concrètement que sur deux ou trois niveaux. Les autres règles (règles 3 et 4), qui ont autant de symboles des deux côtés de la flèche, n'introduisent pas de division des durées.

On notera les grilles sous forme de séquences d'accords, en séparant les mesures par des barres obliques. Ainsi :

I/IV/I/I7/IV/IV/I/VI7/II7/V7/I/I

correspond à la grille représentée dans le tableau 6.2.

I	IV	I	I7
IV	IV	I	VI7
II7	V7	I	I

Tableau 6.2. *Une grille de blues*

L'accord VI7 de la huitième mesure peut être obtenu en appliquant la règle $w\ x7 \rightarrow Dx7\ x7$ avec x = II7 à la grille suivante :

I/IV/I/I7/IV/IV/I/I/II7/V7/I/I

Voici la dérivation complète proposée par Steedman pour obtenir la grille du tableau 6.2 à partir de l'axiome :

I I7 IV I V7 I (production de l'axiome)

– Règles 1 et 2 (chaque unité se divise en deux mesures, avec ajout d'un IV en deuxième mesure par la règle 2) :

I/IV/I/I7/IV/IV/I/I/V/V7/I/I

– Règle 3a (appliquée à V7) :

I/IV/I/I7/IV/IV/I/I/II7/V7/I/I

– Règle 3a (appliquée à II7) :

I/IV/I/I7/IV/IV/I/VI7/II7/V7/I/I

6.4.2. *Moteur d'application des règles*

Dans cette section et la suivante, nous décrivons un petit programme en Common Lisp qui implémente un sous-ensemble de la grammaire de Steedman dans

le sens de la génération [CHE 03]. Le programme comporte un moteur d'application des règles, qui enrichit progressivement une grille de départ (donnée par l'axiome) par application successive des règles, et un « arrangeur » de grille, présenté au paragraphe suivant, qui réalise pour piano solo la séquence harmonique obtenue en fin de processus, avec quelques fragments de style boogie-woogie.

Le principe du moteur d'application des règles est de tirer au hasard une règle, de chercher une position dans la grille pour l'appliquer, puis d'effectuer la substitution. Si la règle tirée n'est pas applicable, on en tire une autre. On recommence ainsi l'opération un nombre de fois fixé à l'avance, ou bien on s'arrête si aucune règle n'est applicable.

La grammaire a été modifiée pour contrôler le découpage de la grille en mesures, en introduisant des « / » dans l'énoncé des règles. Les règles 1 et 2 sont modifiées de la manière suivante :

$$x \rightarrow x\ x$$

est remplacé par :

$$/x/ \rightarrow /x\ x/$$

Ainsi, la division minimale de la grille est fixée à la demi-mesure, car les nouvelles règles 1 et 2 ne sont applicables qu'à des mesures non divisées. Les règles 3 et 4 sont dédoublées. La forme originelle :

$$w\ x7 \rightarrow Dx7\ x7$$

est complétée par :

$$w/x7 \rightarrow Dx7/x7$$

La première s'applique à l'intérieur d'une même mesure, alors que la seconde s'applique de part et d'autre d'une barre de mesure.

Pour introduire les barres de mesure dans le processus de dérivation, on remplace l'axiome par :

$$S \rightarrow \text{I/I/I/I7/IV/IV/I/I/V/V7/I/I}$$

en appliquant systématiquement la règle 1 à tous les symboles de l'axiome originel. En se limitant ainsi aux dérivations de ce nouvel axiome, on exclut certaines séquences, ce qui revient à calculer un sous-ensemble du langage défini par la grammaire de Steedman.

Dans le programme en Common Lisp, les règles sont représentées par des couples formés des parties gauche et droite, dans un codage proche de leur énoncé d'origine, et sont stockées dans une liste. Le moteur comporte une fonction qui détecte si le membre gauche d'une règle apparaît dans une grille, et une fonction qui effectue le remplacement par le membre de droite en cas de succès. La variable apparaissant dans l'énoncé des règles est également traitée par le programme, qui lui affecte une valeur pendant la phase de détection.

La fonction de substitution prend en argument le nombre d'applications de règles souhaité. Cette valeur ne prend pas en compte la règle 1, qui n'enrichit pas à proprement parler la grille. Il peut arriver qu'aucune règle ne soit applicable, auquel cas le processus s'arrête avant que le nombre souhaité initialement ne soit atteint. Quand le degré d'un accord est modifié, on le marque par une * pour indiquer que l'accord ne doit plus être affecté à la variable *w*. Voici deux exemples d'exécution de la fonction, avec respectivement une et dix applications de règles (autres que la règle 1) :

```
 ? (substitue 1)
((i)/(i)/(i)/(i 7)/(iv)/(iv)/(i)/(i)/((ii *) m7)/(v
7)/(i)/(i))

 ? (substitue 10)
((i)/(i)/((v *) m7) ((v *) m7)/((iv# *)) ((iv *))/(iv)
((viib *))/(iv)/(i)/(i) ((iv *))/((ii *) m7) ((v *))/((i#
*)) ((i *))/(i) ((iv *))/(i))
```

Lorsque l'on augmente le nombre de règles, le processus de substitution arrive à saturation après une vingtaine de règles environ. Dans l'exemple ci-après, on affiche le nombre de règles effectivement appliquées. Après vingt-trois substitutions, toutes les mesures ont été divisées en deux (sauf la première et la dernière), ce qui rend inopérantes les règles 1 et 2. De plus, tous les accords de septième sont précédés d'un accord marqué * qui ne peut être affecté à la variable *w*, ce qui interdit l'application de la règle 3. La grille obtenue, assez éloignée du blues originel, donne une idée de la richesse maximale que l'on peut obtenir d'une grille après stabilisation du processus :

```
 ? (substitue 100)
1 = Regle3a-11/2 = Regle2-1/3 = Regle3a-21/4 = Regle3a-
21/5 = Regle1-1/6 = Regle1-3/7 = Regle4-11/8 = Regle2-
1/9 = Regle2-1/10 = Regle4-21/11 = Regle2-3/12 = Regle2-
2/13 = Regle2-1/14 = Regle1-1/15 = Regle4-2/16 = Regle1-
2/17 = Regle3b-11/18 = Regle3a-2/19 = Regle3a-
1/20 = Regle4-31/21 = Regle4-21/22 = Regle3b-
11/23 = Regle4-31/
```

```
(((viib *) m7)/((vi *) m7) ((v# *) m7)/((v *) m7) ((v *)
m7)/((v *) 7) ((iv# *))/(iv) (iv)/(iv) ((viib *))/(i)
((iv *))/((vi *) m7) ((ii *))/((ii *)) ((i# *))/((i# *))
((iv# *))/(i) ((iv *))/(i))
```

VIIbm7	VIm7/V#m7	Vm7	V7/IV#
IV	IV/VIIb	I/IV	VIm7/II
II7/I#	I#/IV#	I/IV	I

Tableau 6.3. *Une grille après saturation des règles*

6.4.3. *Arrangement des grilles pour piano*

Pour écouter une grille produite par le programme décrit au paragraphe 6.4.2, il est nécessaire de la jouer sur un instrument, c'est-à-dire d'en faire un *arrangement* à la façon de logiciels comme *Band-in-a-box*. L'utilisateur saisit une grille d'accords sous forme de chiffrages, et le logiciel calcule un fichier midi dans lequel la grille est arrangée pour les instruments fixés (guitare, basse, batterie, etc.) et dans le style choisi. De tels programmes fonctionnent avec une base de séquences musicales préenregistrées au format midi. Dans le programme Common Lisp présenté ici, le moteur d'application de régles décrit précédemment est complété par un arrangeur de grilles, permettant de réaliser les grilles pour piano solo dans le style boogie-woogie. Les règles traitées dans cette section sont une partie des règles 1, 3 et 4. On a supprimé la règle 2, qui a tendance à perturber le rythme harmonique des grilles comme on l'a dit plus haut, et on a éliminé également les parties des autres règles comportant des accords mineur septième, car la table d'arrangements n'en contient pas.

Formellement, cette composante du programme est une *transduction rationnelle*, au sens défini dans la première partie. Elle prend en entrée une séquence d'accords, et calcule en sortie une séquence de mesures arrangées pour piano. Au cours d'un traitement préliminaire, les accords de la grille sont regroupés en mesures. La succession de mesures ainsi obtenue est ensuite traitée selon le mécanisme de lecture suivant :

– si deux mesures consécutives sont occupées par un même unique accord, on cherche dans la base d'arrangements un motif qui dure deux mesures, et on poursuit la lecture deux mesures plus loin ;

– sinon, on cherche un motif pour la première mesure, et on poursuit la lecture en se décalant seulement d'une mesure.

On retrouve le même schéma de fonctionnement que celui de la figure 6.2. L'état 1 est initial. L'état 2 sert à mémoriser l'accord de la dernière mesure lue. Si la mesure suivante a le même accord, on tire un motif de deux mesures et on revient à l'état initial. Sinon, on tire un motif pour la mesure mémorisée précédemment, et on mémorise l'accord de la nouvelle mesure. Il y a en fait *plusieurs* états 2, associés à chacun des accords possibles (comme les états associés aux lettres 2*a* et 2*b* dans la figure 6.2). Dans le programme en Common Lisp, la fonction qui matérialise l'état 2 distingue ces différents états grâce à un argument supplémentaire.

Les deux composantes du programme, moteur d'application des règles et arrangeur de grilles, illustrent le principe de « factorisation » présenté dans la première partie (figure 6.3). Le programme est *factorisé* en deux parties, la partie harmonique réalisée par la grammaire de Steedman qui produit un « squelette » (la grille enrichie par substitutions), et la partie arrangement réalisée par la composante décrite dans ce paragraphe, qui habille ce squelette et produit un fichier midi (la séquence pour piano solo). La situation est la même que dans les programmes de Barbaud et Cope.

Dans le programme en Common Lisp reproduit en annexe, la fonction d'arrangement est appliquée à une progression harmonique engendrée par juxtaposition de plusieurs grilles, dont la première est la grille de base du blues (celle de l'axiome de Steedman), suivie d'enrichissements progressifs de celle-ci en cinq étapes réalisant quatre substitutions chacune. On peut bien sûr imaginer que l'utilisateur paramètre lui-même ce processus, en choisissant le nombre de grilles produites, et le nombre de substitutions passant d'une grille à la suivante, qui mesure la « vitesse » d'enrichissement.

La séquence harmonique obtenue est ensuite traitée par la transduction qui effectue des tirages dans une table d'arrangements contenant des motifs de piano, qui ont été échantillonnés en les jouant sur un clavier midi. Leur longueur est de une ou deux mesures. Dans la table, les motifs sont codés au format de l'environnement *Open Music* développé à l'Ircam par l'équipe de Gérard Assayag [ASS 97]. Les motifs sont représentés par quatre listes : hauteurs, dates absolues, durées, vélocités, les hauteurs et vélocités étant exprimées en valeurs midi, les dates et durées en multiples de la croche ternaire prise comme unité. Le générateur produit un quadruplet de quatre listes donnant les hauteurs, dates, durées et vélocités de l'ensemble de la séquence calculée. Pour la jouer dans *Open Music*, il suffit d'introduire ces quatre listes dans les entrées d'une boîte *ChordSeq* après avoir effectué quelques réglages :

– les hauteurs sont multipliées par 100 (midicents) ;

– les dates et durées sont converties en millisecondes par multiplication par un facteur 138 (correspondant à la durée de la croche ternaire à un tempo de 145 à la noire pointée : 138 = 60 000/(145 * 3)).

Les motifs échantillonnés dans la table d'arrangement peuvent subir deux transformations : transposition de hauteur et diminution de vélocité. La première permet d'adapter les motifs aux degrés de la grille, chaque motif étant enregistré avec un degré fixe indiqué dans la table. Les motifs correspondant à une mesure divisée en deux peuvent subir une transposition différente pour chaque moitié. La seconde transformation concernant les vélocités permet de rendre l'interprétation moins mécanique, plus vivante.

La table est conçue pour ne contenir qu'un nombre minimal de motifs. La grille est jouée dans la tonalité de Mi majeur. Tous les accords (majeur ou septième de dominante) sont réalisés comme septième de dominante (ce qui correspond au chiffrage 7' de Steedman évoqué plus haut). On a défini deux positions d'accords, l'une avec la tierce en haut pour les degrés V# à I (de Do à Mi), l'autre avec la septième en haut pour les degrés I# à V (de Fa à Si). La distinction de ces deux registres permet de réaliser l'harmonie de manière satisfaisante tout en gardant la table très compacte. Pour chaque accord, on choisit dans la table celui qui appartient au même registre, et on transpose les données midi en conséquence. La figure 6.8 présente en notation musicale une réalisation d'un chorus de blues.

6.4.4. *Analyse par automate fini avec Lex*

On s'intéresse dans ce paragraphe à l'implémentation de la grammaire de Steedman dans le sens de l'*analyse*, c'est-à-dire comme un programme capable de prendre une grille en entrée, et de décider si c'est un blues au sens de Steedman ou non. Le problème est que la grammaire de Steedman n'est ni algébrique, ni régulière. Exceptées les règles 1 et 2, toutes les autres règles comportent plusieurs symboles en partie gauche, contrairement à la condition définissant les règles de type 2 (et de type 3). Or, on a vu que seules les grammaires de type 2 et 3 sont équivalentes à des classes d'automates facilement implémentables (à pile ou finis).

Le problème peut être contourné par une analyse plus fine de la forme de cette grammaire, et par quelques hypothèse restrictives. Les règles contextuelles de la grammaire de Steedman (règles 3 et 4) ont une caractéristique essentielle : elles ont toujours *le même nombre de symboles* à gauche et à droite. Elles ont donc la propriété de conserver la longueur des séquences. Appliquées à un ensemble fini de séquences dont la longueur est bornée, elles produisent un ensemble dont la longueur est également bornée, donc ayant, lui aussi, un nombre fini d'éléments. Cette propriété de finitude montre que l'on peut décrire cet ensemble par une grammaire *régulière*.

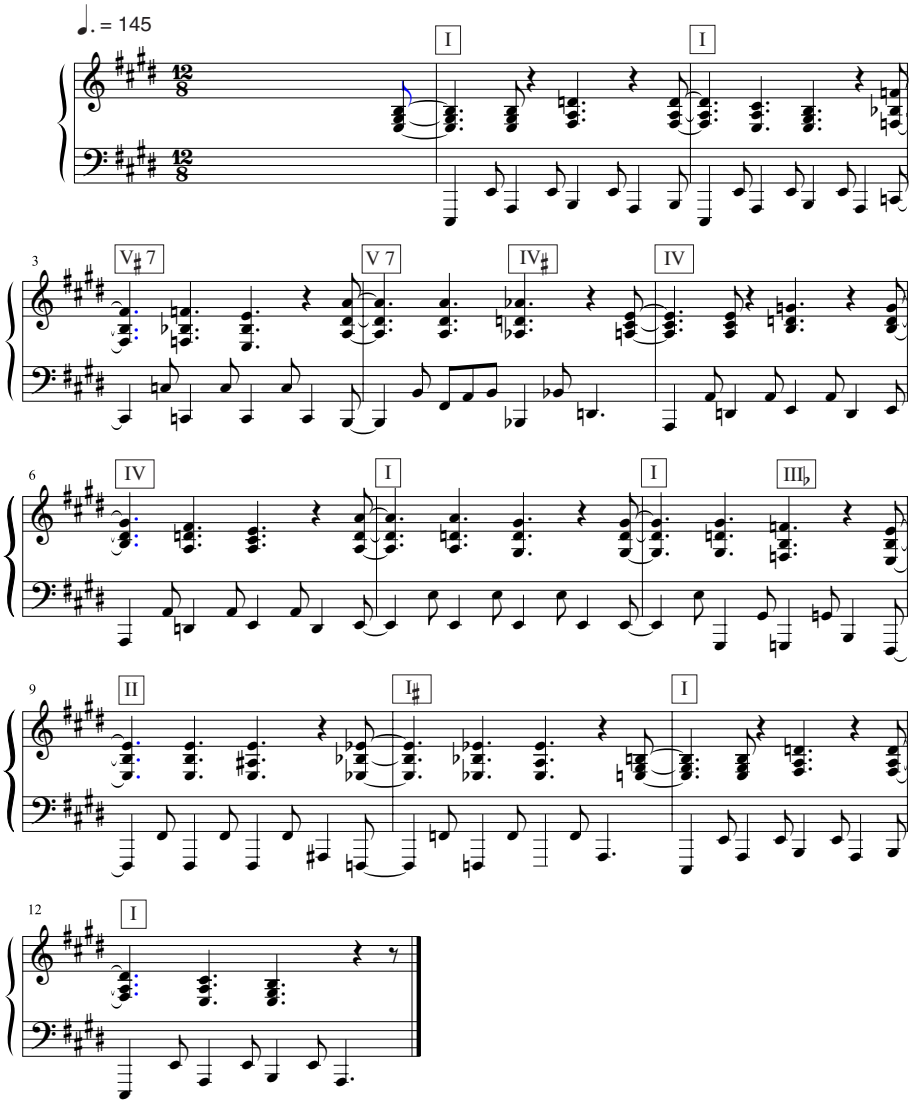


Figure 6.8. *Arrangement pour piano solo d'un chorus de blues*

Il suffit alors de restreindre l'action des autres règles (règles 1 et 2) pour que la grammaire de Steedman puisse être remplacée par une grammaire régulière. Cette restriction appliquée aux règles 1 et 2 peut être justifiée par le fait que les grilles sont rarement divisées au-delà de la demi-mesure. Plus précisément, la grille du

blues n'ayant que douze mesures, une dérivation qui n'introduit pas de quart de mesure ne peut pas comporter plus de douze applications des règles 1 ou 2.

On peut donc construire un automate fini reconnaissant un sous-ensemble du langage engendré par la grammaire de Steedman, ce sous-ensemble étant plus précisément l'intersection du langage engendré par cette grammaire avec le langage des grilles sans division en quart de mesures. On présente ici la construction d'un tel automate avec la commande Lex du système UNIX, en introduisant quelques simplifications supplémentaires dans les restrictions imposées aux règles 1 et 2. On suppose que la règle 1 n'est appliquée qu'en début de dérivation, aux six symboles donnés par l'axiome (pour fabriquer des mesures), puis aux quatre première mesures (pour fabriquer des demi-mesures uniquement dans la première ligne de la grille). Cela revient à limiter la génération aux séquences obtenues en appliquant les autres règles (2, 3 et 4) à la séquence :

I I/I I/I I/I I7/IV/IV/I/I/V/V7/I/I

Quant à la règle 2, nécessaire pour expliquer certains enchaînements de m7 à 7 (comme on le verra ci-après dans l'analyse de *Blues For Alice*), on limite son action en supposant qu'elle n'intervient qu'en fin de dérivation, seulement sous forme de réécriture d'un accord mineur septième.

La construction de l'automate s'effectue de proche en proche, à partir d'une chaîne reconnaissant la séquence ci-dessus. Les états sont les accords, et les flèches indiquent une transition d'un accord à un autre. Pour toute règle applicable à cette séquence (la règle 3a par exemple, I I7 $\rightarrow$ Vm7 I7, applicable à la quatrième mesure avec x = I), on rajoute dans l'automate les états nécessaires pour que la nouvelle séquence obtenue après substitution soit reconnue par l'automate. Ici, il suffit de rajouter un état étiqueté par l'accord Vm7, avec une flèche partant du I précédent, et une autre allant vers le I7 suivant. Lorsque l'on itère cette construction, il arrive un moment où aucune règle ne donne de nouvelles séquences non déjà reconnues par l'automate, puisqu'il n'y a qu'un nombre fini de séquences. A ce stade, la construction s'arrête.

L'automate est complété par quelques transitions supplémentaires. D'une part dans les quatre premières mesures, on relie un accord sur deux afin de reconnaître les grilles dont ces mesures ne sont pas divisées. D'autre part, on ajoute à la fin de la grille quelques *turnarounds* usuels, car dans la plupart des blues, la grille ne se termine pas sur l'accord I, mais sur l'accord V7 ou sur un enchaînement d'accords semi-cadentiel.

L'implémentation de cet automate avec Lex utilise de nombreuses variables auxiliaires pour représenter les différents accords possibles, ainsi que les étapes de la

construction. Les variables représentant les accords mineur septième peuvent se réécrire par la règle 2, en fin de dérivation. La construction de l'automate procède « à reculons » à partir des accords I7 et V7 des quatrième et dixième mesures comme une sorte de préparation amplifiée de ces deux accords cadentiels. Les variables matérialisant les étapes sont groupées en deux sous-séries, correspondant aux deux portions de la grilles traitées indépendamment (mesures un à quatre, mesures cinq à dix).

Une fois la commande Lex exécutée, et le code C compilé, on obtient une fonction réalisant l'automate. Cette fonction lit une grille, et affiche « reconnu », si c'est un blues reconnu par l'automate, et « non reconnu » dans le cas contraire. Elle est utilisée ici pour analyser quelques-uns des vingt-trois blues du *Charlie Parker Omnibook* (qui contient soixante thèmes de Charlie Parker). Ces blues du répertoire bop conviennent bien au style harmonique défini par la grammaire de Steedman, mais on va voir que malgré l'homogénéité de ce corpus, il apparaît une frontière entre des grilles reconnues par la grammaire et d'autres qui ne le sont pas.

Voici d'abord deux blues simples, *K.C. Blues* et *Now's The Time,* reconnus par la fonction :

```
$ blues
> I7/I7/I7/I7/IV7/IV7/I7/I7/IIm7/V7/I7/I7
Reconnu

$ blues
> I7/I7/I7/I7/IV7/IV7/I7/VI7/IIm7/V7/I7/I7
Reconnu
```

En revanche, la grammaire ne reconnaît pas *Barbados,* qui est pourtant un autre blues aussi simple. Le problème vient de ce que les règles de Steedman n'enrichissent la grille qu'en procédant de proche en proche, à partir des cadences des quatrième et dixième mesures. Le turnaround IIm7 V7 de la deuxième mesure ne peut pas s'expliquer, car il est isolé dans un contexte d'accords non cadentiels, et il ne peut être relié par dérivation à l'accord cadentiel I7 de la quatrième mesure.

```
$ blues
> I7/IIm7 V7/I7/Vm7 I7/IV7/IV7/I7/I/IIm7/V7/I7/IIm7 V7
Non reconnu
Mais la grille devient conforme aux règles si on le
supprime :
$ blues
> I7/I7/I7/Vm7 I7/IV7/IV7/I7/I/IIm7/V7/I7/IIm7 V7
Reconnu
```

Certains problèmes apparaissent du fait de restrictions trop grandes imposées dans notre analyseur à la règle 1, qui ne peut s'appliquer qu'aux quatre premières mesures. Ainsi, la grille de *Au Privave* n'est pas reconnue, alors qu'elle est pourtant conforme aux règles de Steedman. Il suffirait en effet d'appliquer la règle 1 à la huitième mesure pour la diviser en deux moitiés, ce qui permettrait ensuite d'expliquer l'enchaînement IIIm7 VI7, dont le rythme harmonique est plus resserré, grâce à la règle 3 :

```
$ blues
> I7/I7/I7/I7/IV7/IV7/I7/IIIm7 VI7/IIm7/V7/I7/IIm7 V7
Non reconnu
```

Les blues complexes, comme le célèbre *Blues For Alice,* ou *Laird Baird* dont la grille est assez proche, sont reconnus par la fonction, et confirment de ce fait la pertinence de la grammaire de Steedman. Ces deux blues utilisent la longue cascade de substitutions II-V qui prépare l'accord I7 de la quatrième mesure. Le second ne diffère du premier que par les accords des sixième et huitième mesures qui n'ont pas été réécrits par la règle 2 (*Blues For Alice* apparaît ainsi comme une dérivation de *Laird Baird*) :

```
$ blues
> I7/VIIm7 III7/VIm7 II7/Vm7 I7/IV7/IVm7
VIIb7/IIIm7/IIIbm7 VIb7/IIm7/V7/I7/IIm7 V7
Reconnu

$ blues
> I7/VIIm7 III7/VIm7 II7/Vm7
I7/IV7/IVm7/IIIm7/IIIbm7/IIm7/V7/I7 VI7/IIm7 V7
Reconnu
```

En revanche, la grille de *Bloomdigo* n'est pas reconnue, alors qu'il s'agit d'une simplification des deux grilles précédentes (la cascade des quatre premières mesures est supprimée). L'accord IVm7 de la sixième mesure ne peut pas s'expliquer s'il n'est pas inséré dans une suite d'accords mineur septième descendant chromatiquement comme dans les deux exemples précédents (il manque l'accord IIIm7, remplacé par I7, ce qui empêche de relier IVm7 et IIIbm7) :

```
$ blues
> I7/I7/I7/I7/IV7/IVm7/I7/IIIbm7/IIm7/V7/I7/IIm7 V7
Non reconnu
```

On voit que la grammaire de Steedman reconnaît une bonne proportion des grilles de blues de Charlie Parker, mais ne fournit pas un modèle de ce corpus complètement adéquat.

6.5. Automate sériel

6.5.1. *Régularité du langage sériel*

Cette dernière partie est consacrée à la modélisation de la règle de base de la musique sérielle, et à la construction d'un analyseur de séquences sérielles. Elle apporte un complément intéressant aux exemples précédents (Barbaud, Cope, Steedman), dans la mesure où le modèle d'automate fini sous-jacent au phénomène musical étudié ici n'est pas directement apparent. Alors que les substitutions des grilles de jazz s'exprimaient naturellement comme des règles de réécriture, la règle de base de la musique sérielle ne se ramène pas immédiatement aux modèles de grammaires et d'automates envisagés dans ce chapitre ; on va voir dans ce paragraphe que pour y parvenir, on est obligé de procéder à une formalisation plus poussée.

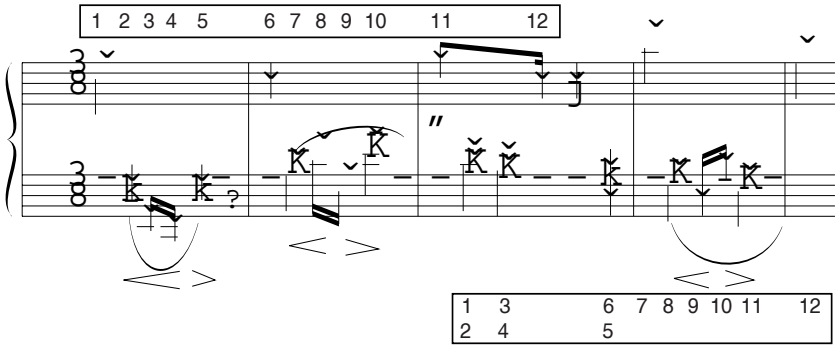


Figure 6.9. *Valse de l'opus 23 de Schoenberg, mesures 29 à 33*

La figure 6.9 est un exemple emprunté à Schoenberg permettant de rappeler la règle sérielle. La pièce est basée sur la série *u = do# la si sol lab fa# la# ré mi mib do fa*, et l'analyse consiste à numéroter les notes dans l'ordre où elles apparaissent dans *u*. Un premier décompte permet de mettre en évidence une première occurrence de la série, correspondant aux numéros 1 à 12 inscrits au-dessus de la portée. On recommence ensuite le même processus avec les notes restantes (notons que l'on peut le cas échéant réutiliser certaines notes déjà numérotées, mais ce n'est pas nécessaire dans cet exemple). Cette deuxième phase montre que les notes restantes constituent à elles seules une deuxième occurrence de la série, associée aux numéros 1 à 12 inscrits au-dessous de la portée. Dire que la séquence est « sérielle », cela revient à dire que l'on peut réitérer le processus de numérotation jusqu'à épuisement de toutes les notes contenues dans la séquence étudiée.

Cette règle de répartition des sons dans l'ordre fixé par une série apparaît historiquement pour la première fois dans cette *Valse* de Schoenberg. A l'époque, celui-ci voulait, entre autres choses, structurer l'univers harmonique atonal qui s'était développé durant les années dix. L'application de la règle sérielle tend en effet, à favoriser certaines configurations d'intervalles qui déterminent la couleur harmonique de la séquence. Plus précisément, les notes simultanées d'une séquence sérielle sont principalement des notes consécutives de la série, ce qui donne à celle-ci un rôle que l'on pourrait qualifier de « réservoir harmonique ». Par exemple, dans la série précédente, les tierces majeures sont favorisées au début de la série (*do# la*) et (*si sol*), puis au milieu sous la forme (*fa# la# ré*), et enfin en bouclant la fin avec le début (*fa la do#*). L'une des configurations préférées des musiciens sériels est l'accord quarte augmentée/quarte juste, qui a été repris abondamment par les pianistes de jazz depuis les années soixante (Herbie Hancock, Chick Corea). La série *fa# fa la sol sib sol# ré do# do mi ré# si* utilisée par Webern dans les *Variations pour piano* op. 27 permet cet accord « fétiche » sous la forme (*ré sol# do#*).

Il est possible de montrer que le langage des séquences construites selon la règle sérielle énoncée ci-dessus est *reconnaissable par automate fini*. Ce résultat a été démontré dans [CHE 90], puis développé dans [CHE 02]. Il ne conduit pas directement à une construction de l'automate qui soit utilisable dans la pratique, mais il repose sur une formalisation de la notion de séquence sérielle (dans sa version simpliste, restreinte à l'énoncé de la règle élémentaire illustrée figure 6.9) à partir de laquelle on peut déduire certaines propriétés caractéristiques. La principale propriété est qu'une séquence est sérielle dans ce sens restreint si et seulement si *toute note apparaissant dans cette séquence est précédée et suivie des notes qui la précèdent et la suivent dans la série*. Cette propriété traduit l'idée intuitive de « plénitude chromatique », qui est implicitement recherchée dans l'utilisation systématique d'une même permutation de douze notes. On peut également formuler cette propriété de manière négative, en disant qu'une séquence n'est pas sérielle si et seulement si il existe une paire de notes (x, y) telle que x précède y dans la série, et telle que l'une des deux conditions suivantes soient vérifiées :

(1) x apparaît dans la séquence, et y n'apparaît pas après x ;

(2) y apparaît dans la séquence, et x n'apparaît pas avant y.

En notant A_x l'ensemble des agrégats qui contiennent la note x, la propriété précédente se traduit sous la forme d'une *expression rationnelle* définissant l'ensemble des séquences non sérielles. En effet, celui-ci est égal à une union d'ensembles qui sont eux-mêmes définis par des expressions rationnelles. Plus précisément, l'ensemble des séquences non sérielles est égal à l'union pour toutes les paires de notes (x, y) telles que x précède y dans la série, des ensembles de la forme $P(x, y) \cup S(x, y)$, où P et Q correspondent respectivement aux conditions (1) et (2) ci-dessus, et sont définis par les expressions rationnelles suivantes :

$$P(x,y) = A^*\left(A_x \cap \overline{A}_y\right)\overline{A}_y\,^*$$

$$S(x,y) = \overline{A}_x\,^*\left(A_y \cap \overline{A}_x\right)A^*$$

L'expression rationnelle ainsi obtenue permet de construire un automate reconnaissant l'ensemble des séquences sérielles. Effectuons l'union partielle des ensembles $P(x, y)$, en faisant varier x pour y fixé. On obtient l'égalité suivante, où P_y désigne l'ensemble des agrégats qui ne contiennent pas y, mais qui contiennent un prédécesseur de y dans la série :

$$\bigcup_x P(x,y) = A^*\left(\left(\bigcup_x A_x\right)\cap \overline{A}_y\right)\overline{A}_y\,^* = A^* P_y \overline{A}_y\,^*$$

De la même façon, effectuons l'union partielle des ensembles $S(x, y)$, en faisant varier y pour x fixé. On obtient une égalité similaire, où S_x désigne l'ensemble des agrégats qui ne contiennent pas x, mais qui contiennent un successeur de x dans la série :

$$\bigcup_y S(x,y) = \overline{A}_x\,^*\left(\left(\bigcup_y A_y\right)\cap \overline{A}_x\right)A^* = \overline{A}_x\,^* S_x A^*$$

Finalement, l'ensemble des séquences non sérielles est égal à :

$$\left(\bigcup_y A^* P_y \overline{A}_y\,^*\right)\cup\bigcup_x \overline{A}_x\,^* S_x A^*$$

Pour chaque terme apparaissant dans l'expression ci-dessus, il est facile de faire un automate reconnaissant le langage correspondant. Dès lors, on peut en déduire un automate reconnaissant le complémentaire de chacun de ces termes, puis construire l'automate de l'intersection de ces langages, qui donne finalement un automate pour le langage sériel lui-même.

6.5.2. *Programme d'analyse par automate fini*

Dans ce paragraphe, on présente un analyseur capable de détecter si une séquence est sérielle ou non (on suppose ici que la série est donnée, la recherche d'une série associée à une séquence supposée sérielle étant un problème beaucoup plus complexe). Comme on l'a vu au paragraphe 6.5.1, cet analyseur peut être décrit

comme une combinaison d'automates, et il reconnaît une séquence comme sérielle dès lors que celle-ci est acceptée par chacun des automates intervenant dans cette combinaison.

On considère figure 6.10 l'automate reconnaissant le complémentaire du langage défini par l'expression rationnelle suivante, pour une note *y* donnée :

$$A^* P_y \overline{A}_y ^*$$

Pour obtenir un automate déterministe, on introduit le complémentaire M_y de l'union de A_y et P_y, de telle sorte que l'ensemble de tous les agrégats soit égal à l'union de M_y, A_y et P_y. L'automate vérifie que toute note précédent *y* dans la série est bien suivie de *y* dans une séquence sérielle. Il a deux états et fonctionne de la manière suivante :

– tant que la séquence lue ne contient pas de note précédent *y*, on reste dans l'état 1 ;

– si on lit une note précédent *y* sans lire en même temps *y*, on passe dans l'état 2 ;

– dans l'état 2 :

- si on lit *y,* condition nécessaire pour que la séquence soit sérielle, on revient à l'état 1 qui est terminal ;

- si l'on ne lit aucun *y* jusqu'à la fin de la séquence, en bouclant sur l'état 2, celle-ci n'est pas sérielle.

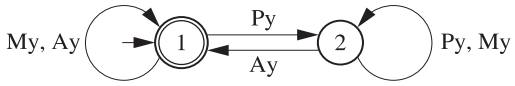


Figure 6.10. *Automate fini contrôlant les notes qui précèdent y dans la série (*P_y*)*

On considère maintenant l'automate représenté figure 6.11, reconnaissant le complémentaire du langage défini par l'expression rationnelle suivante, pour une note *x* donnée :

$$\overline{A}_x ^* S_x A^*$$

Comme précédemment, on introduit le complémentaire N_x de l'union de A_x et S_x, de telle sorte que l'ensemble de tous les agrégats soit égal à l'union de N_x, A_x et S_x, ce qui permet d'obtenir un automate déterministe. Théoriquement, l'automate vérifie

que toute note suivant *x* dans la série est bien précédée de *x* dans une séquence sérielle. Les états 1 et 2 devraient donc être terminaux, mais si l'on ajoute une condition supplémentaire selon laquelle une séquence sérielle contient au moins une fois chacune des notes *x*, cela revient à rendre l'état 1 non terminal. Notons que les états 2 et 3 sont des « puits » dont on ne peut sortir après y être entré, et dans lesquels on peut arrêter la lecture. Le fonctionnement est le suivant :

– tant que la séquence lue ne contient ni *x*, ni une note suivant *x*, on reste dans l'état 1 ;

– si on lit *x*, la lecture s'arrête car toute note apparaissant ultérieurement sera précédée de *x* (en particulier celles qui suivent *x* dans la série, condition pour que la séquence soit sérielle) ;

– si on lit une note suivant *x* sans avoir lu *x*, la lecture s'arrête également, car la séquence n'est pas sérielle.

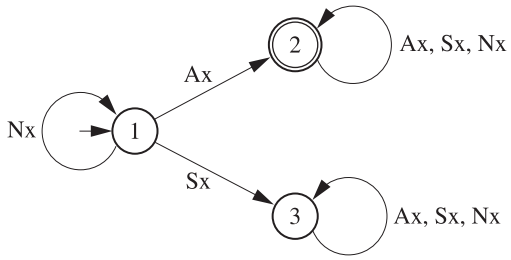


Figure 6.11. *Automate fini contrôlant les notes qui suivent x dans la série (*S_x*)*

L'analyseur reconnaît une séquence comme sérielle si et seulement si pour chaque note de la série, cette séquence est acceptée par les automates des figures 6.10 et 6.11 (on fait l'intersection des langages correspondants). Malheureusement, on ne peut construire avec Lex l'automate résultant de cette combinaison d'automates associés aux douze notes de la série, car cela supposerait un calcul explicite de la table de transition. Or, celle-ci est trop grande pour que le calcul soit envisageable, le nombre d'agrégats possibles (donc d'étiquettes sur les flèches) dépassant les capacités de Lex. On est ainsi obligé d'implémenter l'automate de façon dynamique, en faisant fonctionner séparément chacun des automates associés aux douze notes de la série. C'est ce qui est réalisé ici dans un programme en Common Lisp utilisé pour analyser deux passages comportant des anomalies, dans la *Valse* de l'opus 23 de Schoenberg, construite sur la série indiquée plus haut *u = do# la si sol lab fa# la# ré mi mib do fa*.

L'analyseur détecte une anomalie dans la partition aux mesures 13 à 17, où il manque un *ré* dans le deuxième énoncé de la série, avant l'accord (*mib mi*) :

```
 ? (seriel ? '(reb la si sol lab fa# sib re mi mib (reb
do) fa la si (lab sol) solb sib (mib mi) do fa))
Il manque un ré en partant de la fin
nil
```

Remarquons que l'anomalie n'est plus détectée si l'on prolonge l'extrait analysé par un troisième énoncé de la série jusqu'à la mesure 21. Cet énoncé supplémentaire apporte le *ré* manquant, et la séquence devient sérielle :

```
 ? (seriel ? '(reb la si sol lab fa# sib re mi mib (reb
do) fa la (re si) (lab sol) solb sib (mib mi) do fa do#
la si sol lab solb sib re mi mib (fa do) do# la (sol si
lab solb) (sib re) (mi mib) (fa do)))
La séquence est sérielle
t
```

Il s'agit là d'une faiblesse de l'analyseur, qui n'est pas capable de contrôler la « proximité » relative des notes constituant un énoncé de la série. On ne peut donc analyser la pièce avec un seul traitement. Pour que l'analyseur fonctionne de manière pertinente, il faut l'appliquer à des segments relativement courts, regroupant des énoncés apparemment complets de la série. L'analyseur se charge alors de vérifier qu'aucune note ne manque dans ces énoncés, et qu'elles sont bien disposées dans l'ordre requis.

Une autre anomalie de la partition de Schoenberg se trouve dans l'extrait qui va du deuxième accord de la mesure 63 jusqu'au dernier accord de la mesure 65. Mais dans la segmentation ainsi définie, l'anomalie est masquée par le dernier énoncé de la série qui est tronqué. A la mesure 65, en effet, l'extrait se termine par les deux premières notes d'un énoncé incomplet interrompu avant le *si* :

```
 ? (seriel ? '((la do#) (sol# si sol) (re fa# la# mi)
(do# fa do mib) (la si sol) (lab re fa# sib) (mib fa dob
mi) (la do# sol si) (lab re fa# sib) (la reb fa do mib
mi)))
Il manque un si en partant de la fin
nil
```

Pour faire apparaître l'anomalie, il faut éliminer cet énoncé incomplet, en supprimant les trois derniers accords, de telle sorte que l'extrait se termine exactement à la fin de la série. Dans ce nouvel extrait, délimitant en principe deux occurrences de la série, l'analyseur détecte qu'il faut un *do* à la place du *dob* dans le dernier accord (*mib fa dob mi*) :

```
 ? (seriel ? '((la do#) (sol# si sol) (re fa# la# mi)
(do# fa do mib) (la si sol) (lab re fa# sib) (mib fa dob
mi)))
Il manque un do en partant de la fin
Nil
```

Cet analyseur rudimentaire basé sur des automates finis réalise correctement la tâche consistant à détecter si une séquence est sérielle dans certains cas simples comme ceux correspondant aux deux anomalies étudiées précédemment dans des segments découpés de façon *ad hoc*. C'est le problème de cet analyseur, qui ne fonctionne correctement que sur le plan local, avec des extraits courts comportant des énoncés supposés complets de la série. Cette limite vient de la formalisation de la règle sérielle proposée initialement, qui ne comporte aucun contrôle de la proximité relative des notes constituant un énoncé de la série. La prise en compte de ces conditions de proximité serait nécessaire pour parvenir à une analyse sérielle plus fine. Mais il ne semble pas que les théories analytiques développées autour des musiques atonales et sérielles (notamment dans la musicologie américaine) n'aient jamais résolu de façon satisfaisante ces questions de segmentation et de proximité des notes formant une unité structurelle. Et d'ailleurs, si elles y parvenaient en fournissant une procédure de segmentation explicite basée sur des critères précis, il est probable que leur implémentation nécessiterait un modèle de calcul plus complexe que celui des automates finis.

6.6. Bibliographie

[AHO 89] AHO A., SETHI R., ULLMAN J., *Compilateurs. Principes, techniques et outils*, 1986, InterEditions, Paris, 1989.

[ASS 97] ASSAYAG G., AGON C., Logiciel OpenMusic, Cdrom Forum Ircam, 1997.

[BAR 65] BARBAUD P., *Introduction à la composition musicale automatique*, Dunod, Paris, 1965.

[BAR 68] BARBAUD P., *La musique, discipline scientifique*, Dunod, Paris, 1968.

[BAR 75] BARBAUD P., Ludus margaritis vitreis, rapport Inria, 1975.

[BAR 93] BARBAUD P., *Vademecum de l'ingénieur en musique*, Springer, Paris, 1993.

[BAR 98] BARBAUD P., *Schoenberg*, Editions Main d'Œuvre, Nice, 1998.

[BAR 84] BARONI M., CALLEGARI L. (DIR.), *Musical grammars and computer analysis*, Leo S. Olschki, Firenze, 1984.

[BEC 79] BECKER A., BECKER J., « A Grammar of the Musical Genre Srepegan », *Journal of Music Theory*, vol. 23, 1979, repris dans *Asian Music*, vol. 14, n° 1, 1983.

[BEL 92] BEL B., KIPPEN J., « Bol Processor Grammars », M. Balaban, K. Ebcioglu, O. Laske (dir.), *Understanding Music with AI*, AAAI Press, p. 366-401, 1992.

[BRO 98] BROWN F., « Les trois âges de la musique par ordinateur tels que nous les avons vécus », dans M. Chemillier, F. Pachet (dir.), *Recherches et applications en informatique musicale*, Hermès, Paris, 1998.

[CHE 87] CHEMILLIER M., « Monoïde libre et musique », *RAIRO Informatique théorique*, vol. 21, n° 3 et n° 4, 341-371, 379-417, 1987.

[CHE 88] CHEMILLIER M., TIMIS D., « Toward a theory of formal musical languages », *Proceedings of the ICMC 88*, Cologne, p. 175-183, 1988.

[CHE 90] CHEMILLIER M., Structure et méthode algébriques en informatique musicale, Thèse, Université Paris 7, 1990.

[CHE 90b] CHEMILLIER M., « Langages musicaux et automates : la rationalité du langage sériel », *Actes du colloque Musique et assistance informatique*, Marseille, p. 302-318, 1990.

[CHE 92] CHEMILLIER M., « Automata and music », *Proceedings of the ICMC 92*, San José, p. 370-371, 1992.

[CHE 98] CHEMILLIER M., PACHET F. (dir.), *Recherches et applications en informatique musicale*, Hermès, Paris, 1998.

[CHE 99] CHEMILLIER M., « Générateurs musicaux et singularités », *JIM 99, 6e journées d'informatique musicale*, CNET-CEMAMu, Issy-les-Moulineaux, p. 167-177, 1999.

[CHE 02] CHEMILLIER M., Synchronisation of musical words*, Theoretical Computer Science*, à paraître.

[CHE 03] CHEMILLIER M., Grammaires de substitutions de jazz, à paraître.

[CHO 56] CHOMSKY N., « Three models for the description of language », *IRE Transactions on information theory*, IT-2, 1956.

[CHO 57] CHOMSKY N., *Structures syntaxiques*, 1957, trad. coll. Points, Seuil, Paris, 1969.

[COK 64] COKER J., *Improvising Jazz*, 1964, rééd. Fireside, New York, 1987.

[COP 91] COPE D., *Computers and Musical Style*, A-R Editions, Madison, 1991.

[COP 96] COPE D., *Experiments in Musical Intelligence*, A-R Editions, Madison, 1996.

[COP 99] COPE D., *The Algorithmic Composer*, A-R Editions, Madison, 1999.

[CRO 91] CROSS I., HOWELL P., WEST R. (DIR.), *Representing Musical Structure*, Academic Press, Londres, 1991.

[EIL 74] EILENBERG E., *Automata, languages and machines*, Academic Press, Londres, 1974.

[GRE 73] GREUSSAY P., Modèles de descriptions symboliques en analyse musicale, Thèse de doctorat, Université Paris 8, 1973.

[GRO 69] GROSS M., LENTIN A., *Notions sur les grammaires formelles*, Gauthier-Villars, Paris, 1969.

[GRO 89] GROSS M., PERRIN D. (DIR.), *Electronic Dictionnaries and Automata in Computational Linguistics*, Springer, Berlin, 1989.

[HUG 91] HUGHES D.W., « Grammars of Non-Western Music: A Selective Survey », I. Cross, P. Howell, R. West (dir.), *Representing Musical Structure*, Academic Press, Londres, 327-362, 1991.

[JOH 91] JOHNSON-LAIRD P., « Jazz Improvisation: A Theory at the Computational Level », dans I. Cross, P. Howell, R. West (dir.), *Representing Musical Structure*, Academic Press, Londres, p. 291-326, 1991.

[KLE 56] KLEENE S.C., « Representation of events in nerve nets and finite automata », dans C.E. Shannon, J.M. McCarthy (dir.), *Automata Studies*, Princeton University Press, Princeton, p. 3-42, 1956.

[LAS 73] LASKE O.E., « In Search of a Generative Grammar for Music », *Perspectives of New Music*, p. 351-378, 1973.

[LER 83] LERDAHL F., JACKENDOFF R., *A Generative Theory of Tonal Music*, MIT Press, Cambridge, 1983.

[MAR 13] MARKOV A.A., « Essai d'une recherche statistique sur le texte du roman *Eugène Oneguine* », *Bulletin de l'Académie Impériale des Sciences de Saint-Petersbourg*, vol. 7, 1913.

[MOO 72] MOORER J.A., « Music and Computer Composition », *Communications of the ACM*, vol. 15, n° 2, p. 104-113, 1972.

[MOU 95] MOUTON R., Outils intelligents pour les musicologues, Thèse de doctorat, Université Paris 1, 1995.

[NAT 75] NATTIEZ J.-J., *Fondements d'une sémiologie de la musique*, 10/18, Christian Bourgois, Paris, 1975.

[PAC 98a] PACHET F., « Computer Analysis of Jazz Chord Sequences: Is Solar a Blues ? », dans E.R. MIRANDA (dir.), *Readings in Music and AI*, Harwood, Paris, 1998.

[PAC 98b] PACHET F., CARRIVE J., « Intervalles temporels circulaires et application à l'analyse harmonique », dans M. Chemillier, F. Pachet (dir.), *Recherches et applications en informatique musicale*, Hermès, Paris, p.17-30, 1998.

[PAC 98c] PACHET F., « Sur la structure algébrique des séquences d'accords de jazz », *JIM 98, 5e journées d'informatique musicale*, LMA Marseille, La Londe-les-Maures, 1998.

[PAC 99] PACHET F., « Surprising Harmonies », *JIM 99, 6e journées d'informatique musicale*, CNET-CEMAMu, Issy-les-Moulineaux, p. 187-206, 1999.

[PAR 78] PARKER C., *Charlie Parker Omnibook*, Atlantic Music Corp., New York, 1978.

[PER 93] PERRIN D., « Les débuts de la théorie des automates », *Séminaire Philosophie et Mathématiques*, ENS, janvier 1993.

[RAH 93] RAHN J., « Le compositeur et ses AMIs », *Les cahiers de l'Ircam*, n° 3, 1993.

[RIO 79] RIOTTE A., Formalisation des structures musicales, Polycopié, Université Paris 8, 1979.

[ROA 79] ROADS C., « Grammars as Representations for Music », *Computer Music Journal*, vol. 3, n° 1, 1979, p. 48-55, repris dans C. Roads, J.Strawn (dir.), *Foundations of Computer Music*, MIT Press, Cambridge, 1985.

[ROA 84] ROADS C., « An overview of music representations », dans M. Baroni, L. Callegari (dir.), *Musical grammars and computer analysis*, Leo S. Olschki, Firenze, p. 7-37, 1984.

[ROA 96] ROADS C., *The Computer Music Tutorial*, MIT Press, Cambridge, 1996.

[RUW 71] RUWET N., *Langage, musique, poésie*, Seuil, Paris, 1971.

[SCH 93] SCHWANAUER S., LEVITT D. (DIR.), *Machine Models of Music*, MIT Press, Cambridge, 1993.

[SNE 85] SNELL J.L., « Musical Grammars and Computer Analysis: A Review », *Perspectives of New Music*, vol. 23, n° 2, p. 220-234, 1985.

[STE 84] STEEDMAN M.J., « A Generative Grammar for Jazz Chord Sequences », *Music Perception*, vol. 2, n° 1, p. 52-77, 1984.

[STE 96] STEEDMAN M.J., « The Blues and the Abstract Truth: Music and Mentals Models », dans A. Garnham, J. Oakhill (dir.), *Mentals Models in Cognitive Science*, Erlbaum, Mahwah, 1996.

[SUN 70] SUNDBERG J., LINDBLOM B., « Towards a generative theory of melody », *Swedish Journal of Musicology*, vol. 52, p. 71-88, 1970.

[SUN 91] SUNDBERG J., LINDBLOM B., « Generative Theories for Describing Musical Structure », dans I. Cross, P. Howell, R. West (dir.), *Representing Musical Structure*, Academic Press, Londres, p. 245-272, 1991.

[WIN 68] WINOGRAD T., « Linguistics and the Computer Analysis of Tonal Harmony », *Journal of Music Theory*, 1968, repris dans S. Schwanauer, D. Levitt (dir.), *Machine Models of Music*, MIT Press, Cambridge, 1993.

[XEN 63] XENAKIS I., « Musique stochastique markovienne », dans *Musiques formelles*, 1963, Stock, Paris, p. 61-131, 1981.

Chapitre 7

Programmation visuelle pour la composition

7.1. Introduction

La composition assistée par ordinateur (CAO) est associée, dès ses débuts, à l'histoire de l'informatique musicale. Les premières tentatives de Hiller, Xenakis et autres pour utiliser l'ordinateur en musique dans les années cinquante, adoptaient une approche symbolique (par opposition à l'analyse et la synthèse du signal audio) généralement considérée comme caractéristique de la CAO. La large diffusion, au cours des années quatre-vingt, de nouvelles technologies dans les langages informatiques (langages à objets, programmation logique, langages multiparadigmes) et dans les interfaces graphiques, a conditionné l'émergence d'une CAO « moderne » dans laquelle les concepts liés à la composition et ceux liés à la recherche informatique se sont quelques fois côtoyés de près. En effet, de la même façon que le choix d'un langage de programmation influence le programmeur dans les représentations qu'il favorise ou défavorise, il joue un rôle dans la formalisation d'une idée musicale, et peut susciter des expressions formelles qui ne seraient pas aisées dans un autre contexte. L'utilisation d'un langage de programmation bien défini oblige le musicien à réfléchir sur le processus même de la formalisation et lui évite de laisser l'ordinateur lui imposer ses choix.

Cette CAO contemporaine a bénéficié de nombreuses contributions parmi lesquelles nous citerons Pope [POP 91] qui a proposé plusieurs systèmes basés sur le langage Smalltalk, ouvrant la possibilité pour le compositeur de définir ses propres

---

Chapitre rédigé par Gérard ASSAYAG et Carlos AGON.

objets et méthodes et de bénéficier d'une interface graphique de haut niveau ; Taube [TAU 91] qui a réalisé à Stanford le programme de composition par patterns *Common Music* ; Pachet qui met en œuvre objets et contraintes dans MusES [PAC 94] ; Cointe [ROC 85] qui a proposé le langage Formes, dans lequel les processus musicaux sont représentés par des objets dont l'ordonnancement temporel est contrôlé par des abstractions appelées moniteurs ; Assayag et Malherbe [ASS 85, ASS 86] qui ont réalisé le premier environnement de CAO utilisé à l'Ircam, Crime, basé sur LeLisp de l'Inria, et dans lequel des sous-langages enchâssés permettaient de définir une algèbre compositionnelle avec des sorties imprimées en notation musicale ; Courtot, [COU 90], qui a réalisé Carla, un langage visuel de programmation logique basé sur Prolog II, dans lequel l'utilisateur définit des types d'objets musicaux par combinaisons de types existants ; Laurson, Duthen et Rueda [LAU 96], qui ont mis en œuvre PatchWork, un langage visuel fonctionnel basé sur Lisp, et dont une idée particulièrement originale était la combinaison de graphes d'appel fonctionnels avec des éditeurs graphiques musicaux permettant de visualiser n'importe quel point du calcul.

Cet article a pour objectif de décrire le langage OpenMusic, un langage visuel de CAO récemment mis au point à l'Ircam par les auteurs, aujourd'hui largement utilisé par la communauté internationale des compositeurs et des musicologues et disponible en OpenSource sur SourceForge. Le langage OpenMusic (OM) a pour objectif l'intégration des modèles fonctionnels et objets[1] dans un cadre de programmation visuelle. OM compile vers un langage sous-jacent de haut niveau, Common Lisp/CLOS [STE 98]. Ce choix se justifie par le fait que CL/CLOS est un langage fonctionnel et objet puissant, portable, bien défini, disposant d'un modèle formel solide [CAS 98] articulé autour du concept de *fonction générique*. De plus, une grande quantité de savoir-faire en informatique musicale, notamment à l'Ircam, est associée à Lisp en général [DEH 94]. Enfin, ses qualités de réflexivité et l'existence d'un protocole de méta-objets [PAE 93] facilitent l'implémentation de l'extension visuelle que constitue OM.

La définition d'OM sera donc concentrée autour de la spécification d'une syntaxe et d'une sémantique visuelles pour ce qui concerne le langage et d'un *framework* objet pour ce qui concerne les aspects musicaux. OM a été conçu pour la composition musicale, mais il reste que ce langage visuel ne comporte pas de restriction par rapport à CL/CLOS, et peut donc être utilisé pour la programmation en général et en particulier la pédagogie des langages fonctionnels et objet. Notons aussi que la définition formelle du langage visuel permet d'envisager aisément sa compilation vers d'autres langages disposant de métaprotocole.

1. Le projet initial comportait aussi l'intégration du paradigme de résolution de contraintes. Ce travail est en cours d'élaboration et ne sera pas décrit ici.

Nous décrirons tout d'abord quelques modèles de langages à objet et de langages visuels qui ont été source d'inspiration pour notre travail, puis nous donnerons une partie de la spécification syntaxique et sémantique d'OM, avant de présenter des exemples d'utilisation musicale.

7.1.1. *Modèles de langages à objet*

La définition de la sémantique des langages par objets reste un problème délicat. Cardelli et Wegner ont proposé une extension du lambda calcul typé [CAW 85] dans laquelle les constructions fondamentales sont le polymorphisme et l'abstraction des données. Cardelli et Abadi [ABC 95] proposent une approche plus intuitive dans laquelle les termes primitifs sur lesquels s'effectuent les opérations ne sont pas des abstractions fonctionnelles mais des objets définis comme tels. Tous ces modèles supposent une vision dans laquelle les méthodes font parties de la définition des classes ou des objets, dans la tradition de Simula [JBK 70].

Des langages plus récents tels CLOS ou Dylan utilisent la notion de fonction générique (ou polymorphe, ou à *multidispatching*). La sélection à l'appel d'une fonction se fait dynamiquement sur les types des arguments, vers une méthode spécifique à ces types ou à leurs ascendants. Une différence conceptuelle significative avec la première famille est que l'on considère une fonction, plutôt qu'une classe, comme un ensemble de méthodes. Ce modèle est mieux adapté à nos objectifs parce qu'il rend possible une intégration plus fine des concepts de classe, d'objet et de fonction, et qu'il permet une plus grande « dynamicité », dans le sens où rajouter une méthode ne modifie pas la classe. Notons cependant que la notion de classe perd en modularité, ce qui peut avoir des conséquences sur la réutilisabilité.

Les travaux récents de Castagna [CAS 97, CAS 98] ont permis de proposer un calcul formel, le λ& – calcul, bien adapté aux langages à *multiple-dispatching*. Le λ& – calcul va un peu à l'encontre de l'approche traditionnelle selon laquelle les messages sont passés aux objets. Ici, ce sont plutôt les objets qui sont passés aux messages (les fonctions génériques). Un des grands avantages de ce modèle est la possibilité d'utiliser le *multiple-dispatch*, c'est-à-dire la capacité de sélectionner une méthode en prenant en compte d'autres arguments que le premier argument de la fonction. Une autre caractéristique importante du λ& – calcul est qu'il permet d'ajouter des méthodes à une classe existante sans modifier le type des instances de la classe, ce qui est critique dans le cas d'applications ouvertes comme OpenMusic dans lesquelles l'utilisateur est aussi un programmeur. Le λ& – calcul est donc le modèle formel qui rend le mieux compte de la structure d'OpenMusic.

7.1.2. *Modèles de langages visuels*

On peut classifier les spécifications de langages visuels en trois grandes familles : approches algébrique, logique et grammaticale [MAM 96b]. Nous donnons ici un exemple de chaque approche.

Les langages à icônes basés sur une approche algébrique [CHA 87, CHA 97] utilisent des arrangements d'icônes et d'opérateurs constituant des « phrases visuelles ». Ces phrases peuvent alors être substituées par une nouvelle icône, enrichissant ainsi le vocabulaire. Dans les langages logiques [HAA 95, HAA 96], les termes du langage sont définis par des prédicats en logique descriptive [NAP 98] décrivant les relations topologiques entre entités visuelles. Le langage Pictorial Janus [KSH 91] peut être défini de cette façon. Parmi les avantages de cette approche, on souligne que l'écriture d'un programme est indépendante de l'éditeur, car les spécifications sont d'ordre topologique. Dans l'approche grammaticale [MAM 96a] et [MAM 96b], le langage est défini par les dérivations d'un système de réécriture.

Nous remarquons certaines constantes dans les trois approches étudiées. La plus importante semble être la difficulté de définir les frontières entre lexique et syntaxe. Le choix de primitives lexicales peut donc obéir à différents facteurs, entre autres à la fréquence d'utilisation, au degré de formalisation, etc. Après avoir choisi les primitives, il faut aussi faire des choix sur leur structure ; peuvent-elles avoir des attributs ? Si les primitives possèdent des sous-objets ou des attributs, quel type de calcul est-il permis entre les attributs ? On doit aussi décider de l'interprétation des relations spatiales entre objets et de la structure mathématique sous-tendant les expressions visuelles.

Les spécifications visuelles d'OM, données dans les paragraphes suivants, suivent l'approche grammaticale qui semble la mieux adaptée pour faire le lien entre le formalisme de type λ&–calcul sous-tendant le langage CLOS et le langage visuel d'OM.

7.1.3. *Le langage visuel Max*

Inventé à l'Ircam par Miller Puckette à la fin des années 1980, Max est maintenant pris en charge et commercialisé par David Zicarelli. Le modèle d'exécution de Max obéit à un double modèle : dirigé par les évènements pour ce qui concerne le contrôle et MIDI, à flots synchrones pour ce qui concerne le traitement de signal.

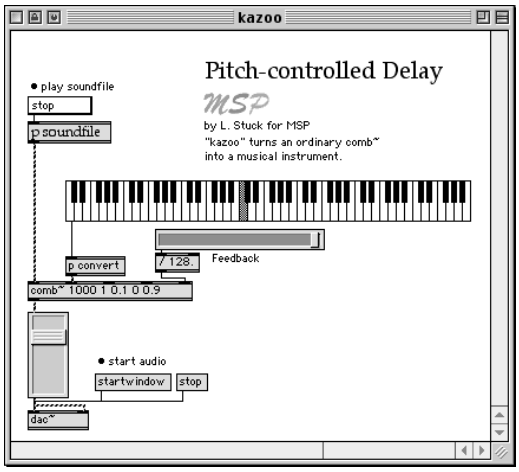


Figure 7.1. *Exemple d'un programme Max*

La métaphore visuelle est celle du studio électronique dans lequel des signaux voyagent à travers des modules et des connections. Le modèle d'évaluation est inversé par rapport à OpenMusic : des messages de contrôle (ou du signal audionumérique) traversent le treillis des connections dans un seul sens (mettons de haut en bas), alors que dans OpenMusic une demande d'évaluation fonctionnelle est propagée de bas en haut, d'une fonction vers ses arguments, et les valeurs retournées redescendent de haut en bas, avec une dynamique qui est celle d'une pile d'exécution. Les avantages de Max sont une extrême efficacité pour le traitement du signal en temps réel, mais l'expressivité en termes de programmation et de structures de données est réduite. C'est donc plutôt un environnement pour la scène. Les avantages d'OpenMusic sont une grande expressivité, un haut niveau de programmation (fonctionnelles, programmation objet, programmation logique, métaprogrammation), mais il n'est pas adapté pour le traitement audio en temps réel, c'est donc plutôt un logiciel pour la composition.

7.2. Le langage visuel d'OpenMusic

7.2.1. *Spécification lexicale*

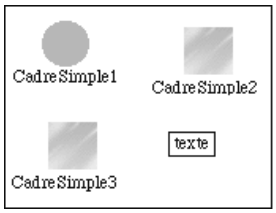
Le choix des primitives lexicales dans cette spécification nécessite un certain nombre de décisions : une description lexicale de très bas niveau rendrait notre description plus formelle mais aussi plus complexe. Nous avons choisi de faire abstraction de la formalisation de certains objets de base et d'assumer leur définition de manière axiomatique. Ainsi, nous supposons l'existence d'un ensemble d'éléments

graphiques de base : l'*icône,* le *rectangle,* la *ligne* et le *texte*. En général, de telles formes ne se rencontrent pas de manière isolée mais sont plutôt combinées pour former des éléments de plus haut niveau que nous appèlerons *primitives lexicales.* Nous supposons, sans entrer dans les détails, que nous disposons de deux types de relations spatiales de base : *contenant* et *en contact*. Les primitives lexicales sont :

– *cadres de texte* : un cadre de texte est défini par un rectangle contenant un texte :

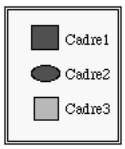
– *cadres simples* : un cadre simple est composé d'une icône en contact avec un *texte* :

– *cadres composés* : un cadre composé est constitué d'un *rectangle contenant* un ensemble de *cadres simples*, de *cadres de texte* et éventuellement un ensemble des *lignes* :



– *lignes* : ligne simple, flèche, ligne pointillée ;

– *séquences* : une *séquence* est une forme particulière de *cadre composé* qui exprime une relation d'ordre entre les *cadres simples* qui lui appartiennent. Cette relation est représentée par la coordonnée des positions des *cadres simples* suivant un des axes du rectangle qui constitue la *séquence* :



Cadre1 < *Cadre2* < *Cadre3*

Un programme en OpenMusic s'exprimera comme un agencement sur l'écran de *cadres composés* vides ou non.

7.2.2. *Spécification syntaxique*

Les entités sont les concepts abstraits de programmation sous-jacents au langage visuel. Elles sont constituées de termes tels que classe, instance, champ, fonction, etc. que nous ne définirons pas et qui sont à considérer dans leur acception habituelle. Chacune de ces entités peut être représentée de manière élémentaire et compacte sous la forme d'une *vue* ou de manière plus développée et éditable sous la forme d'un *container*. Les vues d'une entité sont construites à l'aide de cadres simples et de lignes, ou d'autres vues. Les containers sont des cadres composés, auxquels on adjoindra éventuellement des cadres de textes en contact avec le bord supérieur. Plusieurs vues, mais un seul container d'une même entité, peuvent coexister visuellement à un moment donné.

La notion de base dans OpenMusic est le *patch*. Le patch réifie le concept d'algorithme. Les patchs contiennent des *boîtes* (icônes) et des *connections* entre elles. La figure 7.2 montre un patch en OpenMusic.

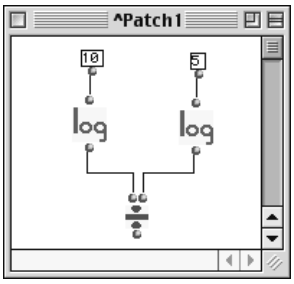


Figure 7.2. *Un patch en OpenMusic*

Les *boîtes* apparaissent exclusivement dans le corps d'un patch, et sont en effet des *invocations* : invocation d'une fonction générique, invocation d'une classe et invocation d'un patch. L'invocation d'une classe résulte par définition, en la création d'une nouvelle instance de cette classe. Un patch est une lambda-abstraction simple, à savoir une fonction non typée en entrée. Une boîte d'invocation de patch apparaîtra donc à l'intérieur du corps d'une méthode ou bien dans le corps d'un autre patch. Les containers associés à ces boîtes comportent entre autres composants, une *référence* qui pointe vers le container de l'entité associée. Par exemple, le composant *référence* du container de la *boîte* d'invocation de fonction générique désigne le container de la fonction générique en question, lequel contient

des vues de méthodes, lesquelles sont associées aux containers de méthodes qui contiennent le code visuel de ces méthodes.

Les *inlets* et les *outlets* des boîtes représentent des entrées et des sorties. L'indice *n* permet de garder trace de l'ordre de ces entrées/sorties. Voici par exemple une vue de *boîte* qui pourrait correspondre à l'invocation d'une fonction générique à *n* arguments et à *m* sorties (figure 7.3).

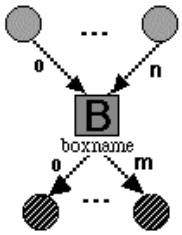


Figure 7.3. *Xxxxxxxxxxxxxxxxx*

7.2.3. *Sémantique opérationnelle*

Le patch est un objet central dans OpenMusic. Dans sa forme développée (container de patch), il constitue pour le langage visuel une *expression* correspondant à la notion commune de *corps de programme*. A l'intérieur d'un patch, il y a des boîtes et des connexions qui représentent un graphe d'invocations fonctionnelles. On peut demander l'*évaluation* en n'importe quel point de ce graphe (sur n'importe quelle boîte). L'évaluation d'une boîte est déclenchée lors de l'évaluation d'une de ses sorties (*outlets*). Il y a ainsi une chaîne d'évaluations qui correspond à l'exécution du programme.

Nous allons donc interpréter les éléments décrits jusqu'ici en termes d'évaluation. La relation d'équivalence définie dans les règles suivantes suppose que les vues multiples d'une même entité sont équivalentes entre elles ; elles sont aussi équivalentes aux containers de la même entité. La sémantique est donnée sous la forme d'un ensemble de règles de réductions qui transforme les expressions graphiques en expressions Lisp. Ces règles utilisent un petit nombre de primitives Lisp telles que *first*, *rest*, *member*, et *funcall*. A titre d'exemple, nous montrons les règles d'évaluation pour une boîte d'invocation de patch, pour l'ensemble complet de règles (voir [AGO 98]).

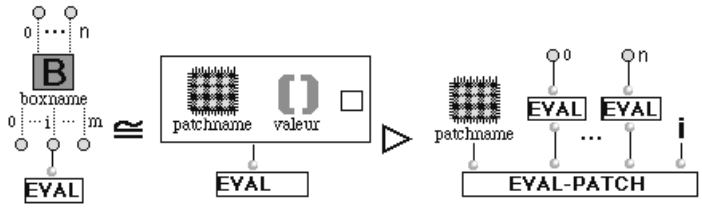


Figure 7.4. *(R8) Boîte invocation de patch*

L'évaluation d'une boîte invocation de patch par son i^e^ *outlet se réécrit en* un appel à Eval-Patch (R11) avec les paramètres suivants : la référence de la boîte, l'évaluation du premier *inlet* de la boîte, … , l'évaluation du n^e^ *inlet* de la boîte et l'indice de l'*outlet* évalué.

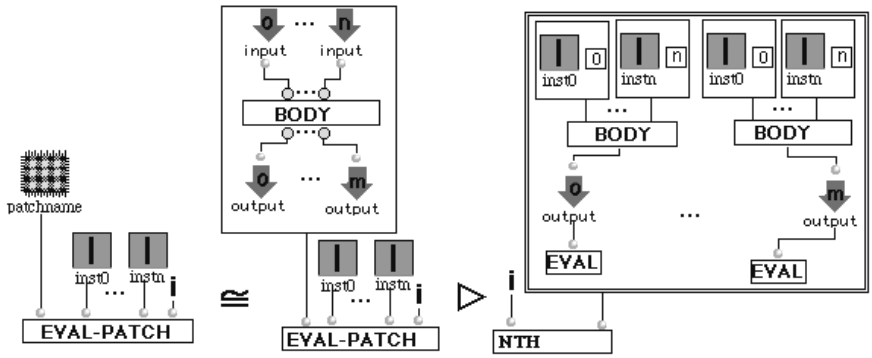


Figure 7.5. *(R11) Patch*

L'évaluation d'un patch (Eval-Patch) avec arguments inst0, … , instn et i *se réécrit en* le i^e^ élément d'une liste (v0 v1 ... vm) correspondant aux évaluations des m + 1 output. Pour tout $input_j$ sa valeur est remplacée par la valeur d'entrée $inst_j$.

7.3. Implémentation

OpenMusic a été implémenté sur le langage CommonLisp/CLOS sur Apple Macintosh. Le système de métaprogrammation de CLOS contient une partie statique consistant en une hiérarchie de classes de méta-objets et une partie dynamique ou protocole de méthodes, qui permet la manipulation des méta-objets. Nous entendons par méta-objets les divers éléments du système objet sous-jacent à CLOS, tels que les fonctions génériques, les méthodes, les variables d'instances (*slots*) et les classes.

Les deux tâches principales du métaprogrammeur sont : définir les sous-classes des classes de méta-objets et définir ou redéfinir des méthodes pour ces nouvelles classes ou celles déjà existantes [PAP 93]. C'est cette technique que nous avons utilisée pour implémenter OpenMusic. OpenMusic constitue donc un enrichissement de la syntaxe et de la sémantique de CLOS par métaprogrammation. Cependant, certaines entités visuelles, telles que patch ou boîte, n'ayant pas de correspondant dans CLOS, ne peuvent être constituées par métaprogrammation (en effet, il n' y a pas de classe permettant de réifier la notion de programme ou d'invocation fonctionnelle dans CLOS). Nous ne ferons pas une description détaillée de CLOS, le lecteur peut consulter [STE 98] et [PAE 93].

Pour implémenter OpenMusic, nous avons décidé d'opérer la même division que CLOS entre une partie statique et une partie dynamique. L'un des problèmes décisifs dans notre conception a été le niveau de détail auquel on définit le protocole ; en général, un protocole très détaillé est peu modulaire ; ainsi les changements doivent être effectués en plusieurs endroits, ce qui affecte la fiabilité. Par contre un protocole peu détaillé rend délicat le choix de l'endroit où introduire les modifications.

7.3.1. *Partie statique*

La figure 7.6 illustre la partie statique d'OpenMusic. Les classes de méta-objets CLOS sont représentées par un rectangle gras.

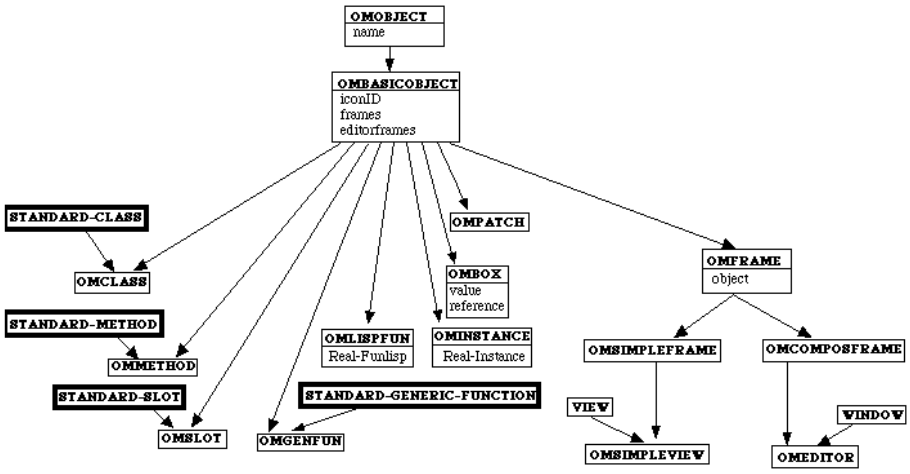


Figure 7.6. *Hiérarchie de classes de méta-objets dans OpenMusic*

Cette hiérarchie de classes rend compte de tous les objets manipulables dans l'environnement visuel et dans le langage de programmation visuel d'OpenMusic. Les classes fondamentales du système objet visuel d'OpenMusic, *OMClass*, *OMMethod*, *OMSlot* et *OMGenFun*, héritent d'une part de la classe *OMObject* et d'autre part d'une classe fondamentale CLOS homologue. Les classes *OMPatch* et *OMBox* réifient respectivement les notions de patch et de boîte inexistantes dans CLOS. La classe *OMLispFun* (qui réifie la notion de fonction Lisp simple, non générique), ne peut être constituée par métaprogrammation, car la classe fondamentale CLOS homologue *function* constitue une primitive qui n'est pratiquement pas sous-classable. Dans ce cas précis, nous opérerons par une sorte de délégation en agissant sur le champ *Real-FunLisp* qui contient un pointeur vers la fonction Lisp. Une dernière classe, *OMInstance*, est introduite afin de visualiser les instances en général. De même que pour les fonctions Lisp, le champ *Real-Instance* d'une *OMInstance* contient un pointeur vers l'instance CLOS. Les huit classes mentionnées, constituent les objets de calcul de notre langage (partie gauche de l'arbre d'héritage). La partie droite de l'arbre d'héritage implémente le paradigme visuel. On peut noter que la classe *OMFrame* hérite aussi de la classe *OMBasicObject*. Cela signifie que le paradigme visuel fait partie du langage, ceci étant fait dans l'intention de permettre à l'utilisateur de modifier et de contrôler ce paradigme.

Tout *OMObject* a un nom (*name*) et peut être visualisé sous deux formes, soit comme une vue simple ou comme un éditeur (équivalent au *container* détaillé plus haut). Un éditeur est une instance de la classe *OMEditor*, tandis qu'une vue est une instance de la classe *OMSimpleView*. Tout objet de visualisation (instance de la classe *OMFrame*) est fortement attaché à un objet de calcul. Ce lien est exprimé par le champ *object* de la classe *OMFrame*. Les entités du langage, à leur tour, possèdent deux champs, *frames* et *editorframes*, qui contiennent respectivement une liste des vues simples et un éditeur qui visualisent l'objet en question.

7.3.2. *Protocole dynamique*

Comme on l'a vu, les objets sont composés par un ensemble d'éléments et optionnellement par des relations entre ces éléments. Par exemple, une classe est une liste ordonnée des champs, une fonction générique consiste en un ensemble de méthodes, un patch contient une liste de boîtes connectées, etc. Une fonction fondamentale dans notre protocole est donc la fonction *get-elements*, celle-ci retournant pour chaque *OMBasicObject* la liste de ses éléments, par exemple pour une OMClass nous aurons :

get-elements : OMClass → liste de OMSlots

Le mécanisme de « glisser-déposer » est le principal outil d'édition dans OpenMusic. Une opération de glisser-déposer est définie comme une action entre une instance de la classe *OMsimpleView* appelée « source » et une instance de la classe *OMEditor* appelée « cible ». Il existe un ensemble de prédicats *allow-drop* qui déterminent si l'opération de glisser-déposer est possible entre une paire (source, cible). Les fonctions *add-element* et *remove-element* ajoutés aux mécanismes de glisser-déposer et de délégation définissent le paradigme d'édition d'objets :

add-element : OMClass × OMSlot → OMClass

remove-element : OMClass × OMSlot → OMClass

Glisser une vue (source) dans la fenêtre d'un éditeur (cible) produit en général l'application de la fonction générique *add-element* aux paramètres cible et source. Cette fonction est chargée d'effectuer les modifications visuelles dans l'éditeur ; elle doit aussi déléguer l'invocation de la fonction *add-element* au couple formé par les champs *object* de cible et de source. Cette dernière invocation est chargée de modifier l'objet de calcul visualisé par l'éditeur. Les fonctions suivantes, qui complètent le protocole, suivent le même paradigme :

rename : Obj × name → Obj
Change le nom d'*Obj* par *name*

change-icon : Obj × iconID → Obj
Change l'icône d'*Obj* par *iconID*

select : Obj → Obj
Sélectionne *Obj*

unselect : Obj → Obj
Désélectionne *Obj*

moveobject : Obj × position → Obj
Change la position d'*Obj* à *position*

connect : SourceBox × i × TargetBox × j → Bool
Connecte la i^e sortie de la boîte *SourceBox* à la j^e entrée de la boîte *TargetBox*

box-value :Box × i → Obj
Evalue la i^e sortie de la boîte *Box*

7.4. L'environnement OM

7.4.1. *Workspaces*

L'interface principale dans OpenMusic est le *workspace*. Le *workspace* est un gestionnaire de fichiers iconiques assurant la persistance de l'environnement, les échanges avec le système hôte se faisant par glisser-déposer. Plusieurs *workspaces* indépendants peuvent être maintenus, cependant un seul *workspace* peut être visualisé sous la forme de container à un instant donné (voir figure 7.7). Les *folders* ou dossiers permettent d'organiser hiérarchiquement l'information.

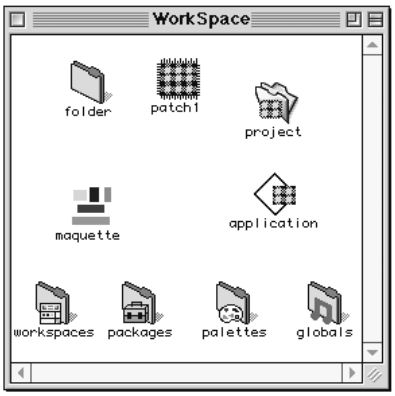


Figure 7.7. *Un container de* workspace

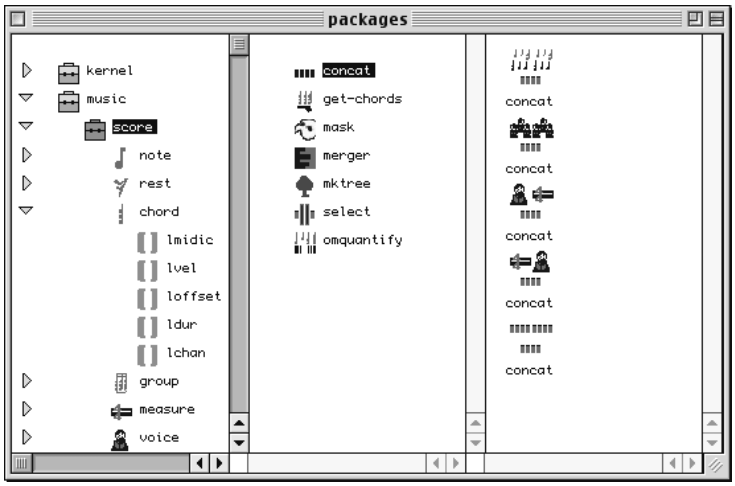


Figure 7.8. *Un container de packages*

7.4.2. *Packages*

Les *packages* sont des dossiers spéciaux qui archivent les classes et les fonctions génériques, à la manière du langage Java. Dans la figure 7.8, le package `music` est ouvert, ainsi que le sous-package `score`. En dessous de `score`, apparaissent les classes disponibles dans ce package. La classe `chord`, ouverte, montre les champs de cette classe. Dans la seconde colonne apparaissent les fonctions génériques appartenant à `score`. Dans la dernière colonne, les méthodes appartenant à la fonction générique `concat`, avec leurs types d'entrées sous la forme de mini-icônes. Un autre éditeur existe pour les packages, permettant de visualiser l'arbre de hiérarchie des classes (voir figure 7.9).

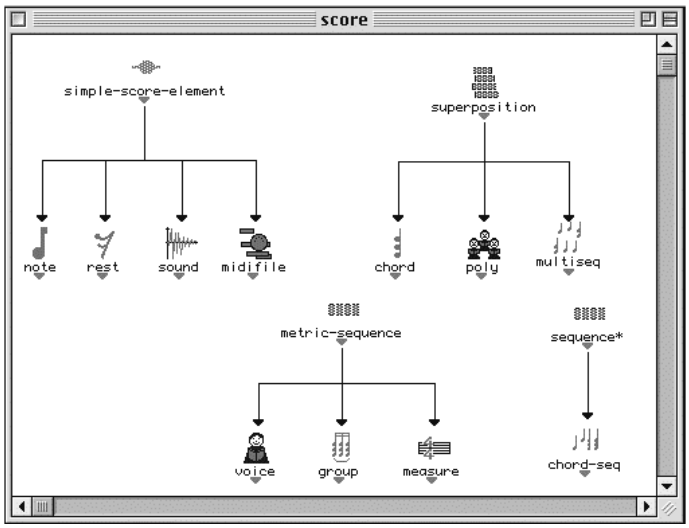
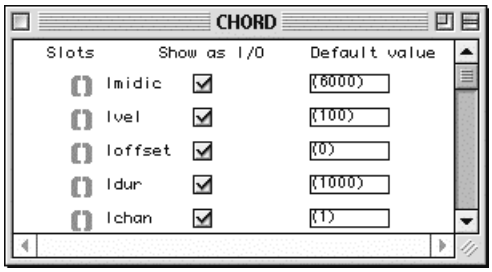


Figure 7.9. *Arbre des classes musicales*

7.4.3. *Classes*

Les packages montrent des vues iconiques de classes. Ouvertes, celles-ci ont un éditeur qui permet de manipuler les champs. Voici, par exemple, l'éditeur de la classe `chord` (figure 7.10).

Les nouveaux champs sont créés en glissant-déposant sur l'éditeur une icône de classe. Un nouveau champ typé par cette classe apparaît alors. Dans la classe `chord`, tous les champs sont de type *list*, indiqués par une icône caractéristique. Les valeurs par défaut sont entrées au clavier, ou bien en glissant-déposant un objet quelconque de l'environnement. L'attribut `show i/o` sera expliqué plus loin avec les *factories*.



Lmidic. Une liste de midicents (hauteur des notes)

Lvel. Une liste de vélocités midi

Loffset. Une liste d'offsets temporels (appoggiatures)

Ldur. Une liste de durées

Lchan. Une liste de canaux midi

Figure 7.10. *Editeur de la classe* `chord`

7.4.4. *Sous-classage*

Les nouvelles classes sont créées dans le package `user`. Nous y plaçons un *alias* de l'icône de classe `chord` vue précédemment. Les alias permettent de créer des liens invisibles entre les différents packages (ici avec le package `score`) et de morceler ainsi les arbres d'héritage entre ces packages. Ensuite, nous créons une nouvelle classe `virchord` et plaçons une flèche d'héritage issue de `chord`.

Dans le container de `virchord`, nous créons un champ `bpf`, en glissant-déposant la classe de même nom (les bpf sont des fonctions linéaires par segment, associées à un éditeur graphique). Nous voudrions que, lors de l'initialisation d'une instance de `virchord`, le champ `bpf` soit rempli avec une nouvelle instance de la classe `bpf`, dont les ordonnées seraient les hauteurs de l'accord. Nous voudrions de surcroît, toujours à l'initialisation, qu'une note grave (appelée fondamentale virtuelle) soit calculée et rajoutée aux notes données pour l'initialisation des champs. Il suffira pour cela de définir visuellement une fonction d'initialisation que nous ne détaillons pas ici.

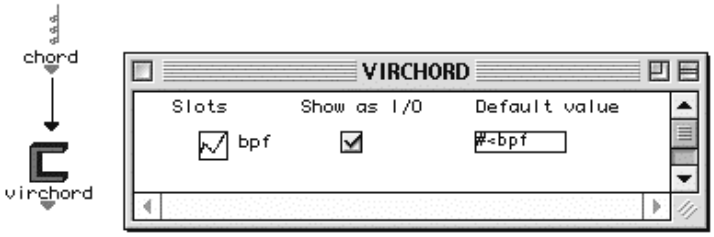


Figure 7.11. *Exemple d'héritage*

7.4.5. *Patches, factories, instances, éditeurs*

Une fois la nouvelle classe définie, nous pouvons créer par glisser-déposer de cette classe sur une fenêtre de patch, une boîte *factory* (figure 7.12 en haut à gauche). Cette factory est en fait une boîte de création d'instances `virchord`. Elle possède tous les *inlets* (*self* + champs) et les *outlets* (self + champs) caractéristiques de la classe `virchord`. Sur l'*inlet* correspondant au champ `lmidic` est connectée une constante contenant une liste de hauteurs en midicents. A l'évaluation de la factory (par clic sur un des *outlets*), le mini-éditeur occupant le rectangle de la factory est mis à jour avec l'accord nouvellement créé. Cet accord est disponible sur la sortie `self`, et ses champs sur les autres sorties, pour de futures connections à d'autres boîtes. Un mode d'évaluation particulier (majuscule-clic) permet aussi de créer une instance qui se matérialise sur l'écran et se sépare de la factory, soit l'équivalent visuel d'une *variable* (boîte en dessous de la factory). A partir de cette instance, un éditeur musical peut être ouvert par double-clic (en dessous de l'instance), ou encore un éditeur objet (à droite). Ce dernier montre la structure stratifiée liée à l'héritage, chaque ensemble de champs étant rangé au niveau de la classe qui le définit. Les valeurs des champs étant des instances, celles-ci peuvent de nouveau être ouvertes par double-clic, éditées dans l'éditeur approprié, changées par glisser-déposer d'une instance venant d'une autre zone de l'environnement, ou encore exportées en les glissant-déposant dans une autre zone.

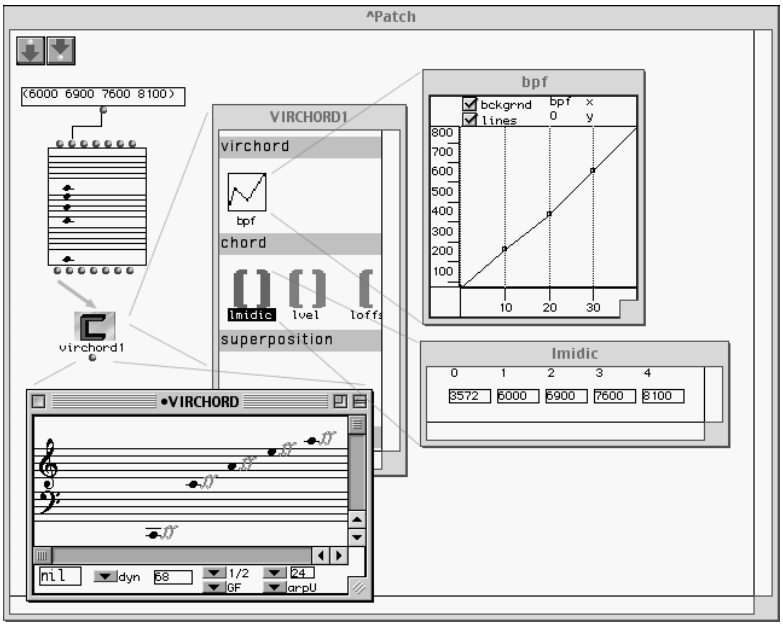
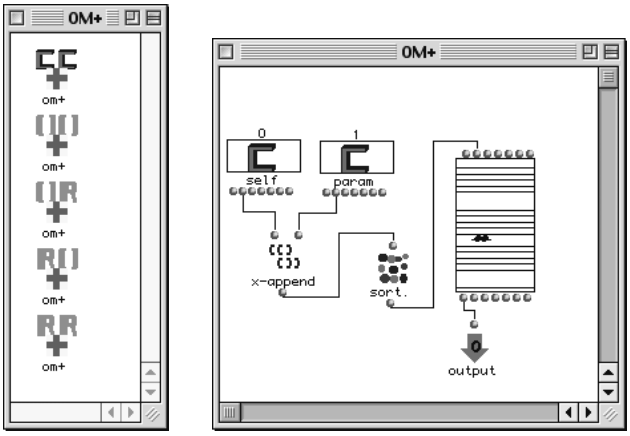


Figure 7.12. *Création et édition d'une instance de* `Virchord`

7.4.6. *Fonctions génériques, méthodes*

La fonction générique prédéfinie `OM+` additionne deux objets de type `nombre` ou `liste` de nombre (effectuant dans ce cas une opération vectorielle). Son éditeur-container (figure 7.13a) montre la collection de méthodes correspondant aux combinaisons de ces types. Nous allons augmenter cette collection en lui ajoutant une méthode capable d'additionner deux accords `virchord` (figure 7.13b). L'icône de la classe virchord est glissée-déposée sur les représentants de classe constituant les entrées de la méthode (en haut à gauche, fenêtre de droite). Ces entrées maintenant typées, présentent tous les champs de `virchord` sous forme d'*outlet*. Les champs `lmidic` des deux accords sont concaténés et retriés en ordre de hauteurs ascendantes, puis envoyés dans l'entrée lmidic d'une factory `virchord`. Une nouvelle instance de cette classe est ainsi créée, dont les hauteurs sont l'union des hauteurs des accords d'entrée. Le container de la fonction générique OM+ est mis à jour avec la vue iconique de la nouvelle méthode, dont les types d'entrée sont indiqués par deux mini-icônes (en haut, fenêtre de gauche).



a) b)

Figure 7.13. *Création d'une méthode*

La figure 7.14 montre un patch dans lequel deux `virchord` sont « additionnés » grâce à la fonction générique `OM+`, qui dispatche dans ce cas vers la nouvelle méthode. Le résultat, une instance de `virchord`, est directement branché dans l'entrée `self` d'une factory de la même classe. Il est à noter que l'utilisation directe de l'entrée `self` inhibe les autres entrées. Cela signifie que l'on désire initialiser la nouvelle instance, non pas par des valeurs de champs, mais par la copie d'une instance existante (mécanisme appelé « prototypage »).

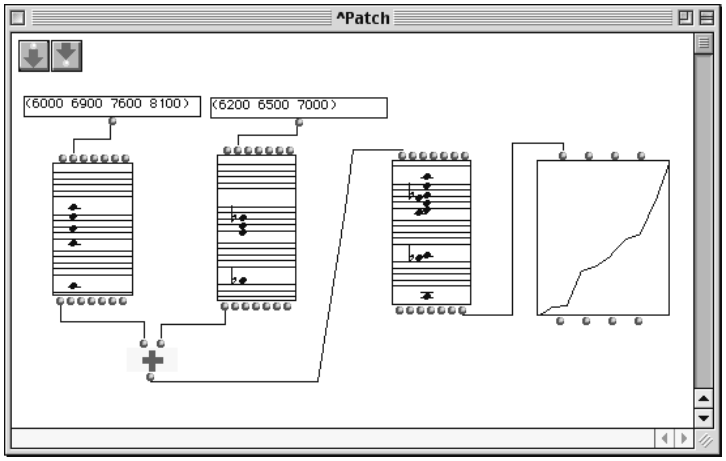


Figure 7.14. *Addition de deux accords*

7.4.7. *Editeurs musicaux*

Les figures suivantes montrent quelques-uns des éditeurs musicaux d'OM. Ces éditeurs sont des containers particuliers qui, au lieu de montrer la collection des composants d'une classe (en l'occurrence une classe musicale telle que `chord` ou `voice`) interprètent ces composants et élaborent une représentation globale, en notation musicale par exemple. D'autres éditeurs du même type peuvent ainsi exhiber une représentation piano-roll ou forme d'onde.

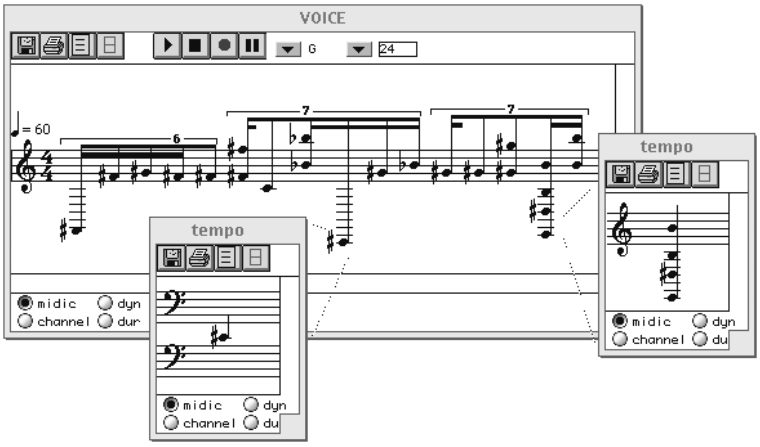


Figure 7.15. *Editeurs de* `voice` *et de* `chord`

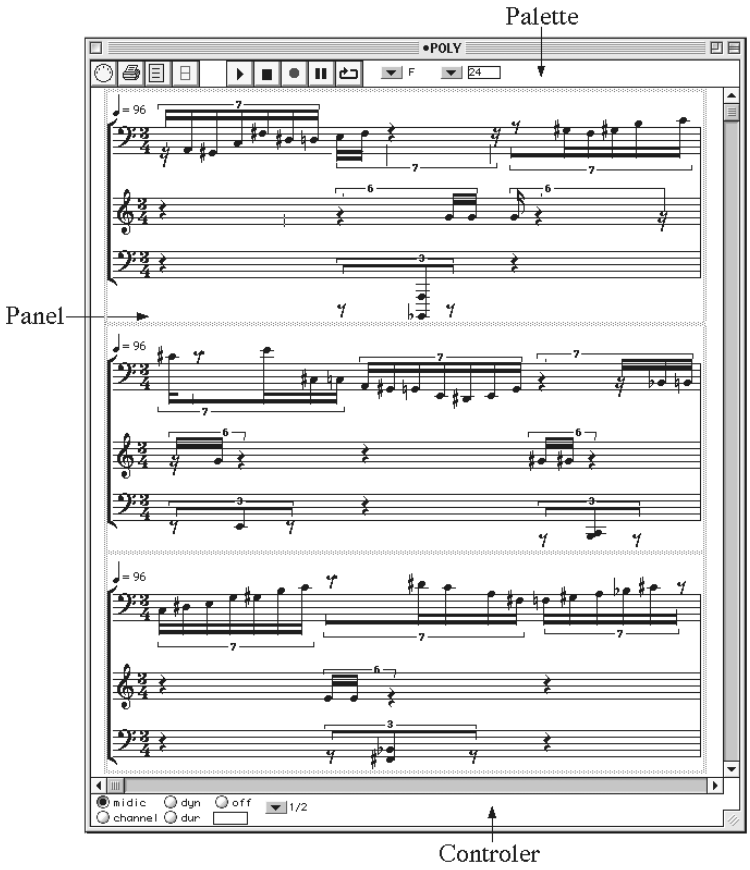


Figure 7.16. *Un éditeur polyphonique*

La figure 7.15 montre un éditeur de la classe `voice`, à partir duquel les accords peuvent être ouverts dans des éditeurs de la classe `chord`. La figure 7.16 montre un éditeur de la classe `poly`.

7.5. Maquettes

Les *maquettes* sont des entités hybrides qui articulent trois concepts :

– la programmation algorithmique des processus musicaux ;

– la structure temporelle, à savoir le positionnement relatif ou absolu des objets musicaux dans le temps ;

– la structure hiérarchique des formes musicales.

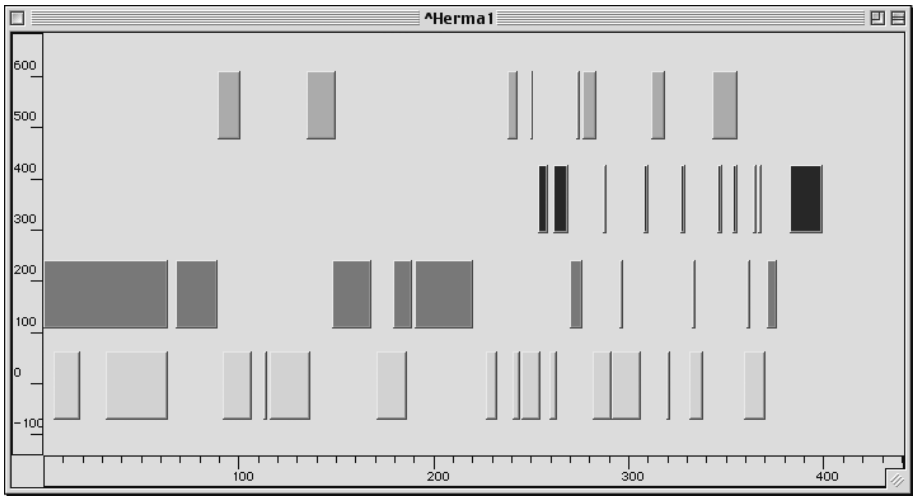


Figure 7.17. *La maquette de Herma*

Les maquettes sont des composants de base d'OM, comme les patches ou les classes. A ce titre, elles apparaissent directement sur le *workspace*. Comme les classes, elles peuvent être déposées dans la fenêtre d'édition d'un patch : elles deviennent alors génératrices d'instances de la classe `maquette` et peuvent alors prendre des arguments, qui servent à les initialiser, et devenir sources de valeur, à travers un *outlet* qui peut être connecté à l'entrée d'autres boîtes. Enfin, elles disposent d'un éditeur, mode d'interaction privilégié pour créer des structures musicales sophistiquées.

La figure 7.17 montre l'éditeur d'une maquette contenant le modèle d'une pièce de Iannis Xenakis, Herma, pour piano[2]. Dans son ouvrage *Musiques formelles* [XEN 91], Xenakis décrit dans le détail le procédé de construction de Herma, ce qui a permis de l'implémenter très simplement sous forme de maquette. La pièce est composée de quatre strates caractérisées par leur intensité, indiquées par des couleurs différentes. Chacune contient un ensemble de blocs distribués dans le temps (la dimension horizontale). Chaque bloc est une séquence musicale contenant un nuage de notes. Il y a deux niveaux de calcul dans Herma :

– les blocs calculent des nuages de notes dont la répartition temporelle obéit à une distribution exponentielle, paramétrée par une densité moyenne d'évènements, caractéristique de chaque bloc. Les hauteurs ainsi distribuées dans le temps sont puisées, elles, dans le réservoir de hauteurs associé au bloc ;

2. Composée de 1960 à 1961, Herma fut créée par le pianiste Y. Takahashi à Tokyo en 1962.

– les réservoirs de hauteurs associés à chaque bloc sont soit des sous-ensembles des quatre-vingt-huit notes du clavier (ensembles A, B et C), choisis au départ par Xenakis selon des critères de complémentarité harmonique, soit des combinaisons ensemblistes des réservoirs associés aux blocs précédents. Au fur et à mesure qu'avance le temps, une expression algébrique se construit, et arrive à terme sur le dernier bloc, qui révèle en quelque sorte le ressort formel de la pièce.

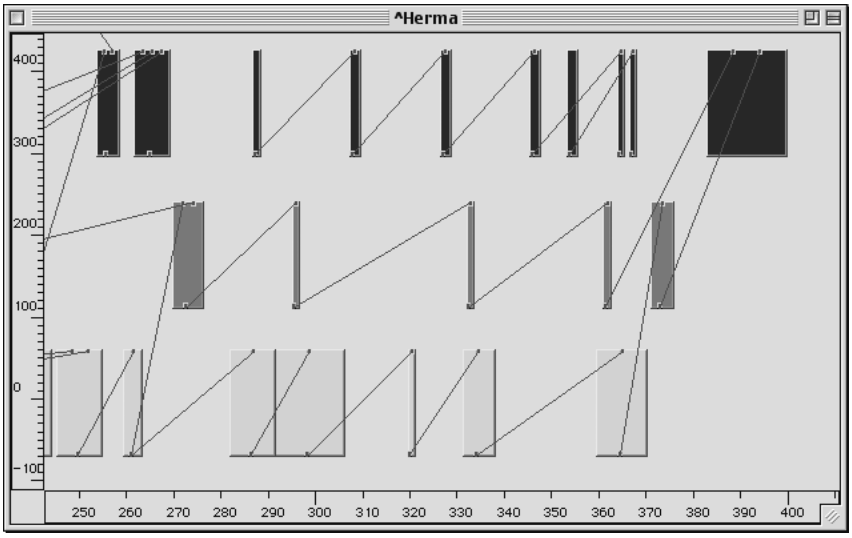


Figure 7.18. *La maquette de Herma, détail des connexions*

L'expression définissant la construction du réservoir final se donne comme suit :

$$\overline{(\overline{A} \cap \overline{B} \cup A \cap B)} \cap \overline{C} \cup (A \cap B \cup \overline{A} \cap \overline{B}) \cap C$$

Cette expression calcule un ensemble formé de l'union des parties propres de A, B et C avec l'intersection des trois ensembles. Chacun des blocs qui précède le dernier en calcule une sous-expression, par exemple $\overline{A} \cap \overline{B}$, l'utilise comme réservoir, puis la passe aux blocs suivants qui la recombinent avec d'autres, et ainsi de suite jusqu'au bloc final. Xenakis joue habilement de ces ensembles de hauteurs pour créer des couleurs harmoniques habillant les nuages stochastiques, les termes intermédiaires formant une palette de contrastes (lorsqu'il superpose des réservoirs sans éléments communs) et de fusions (si, au contraire on ne conserve que les éléments communs). Une option d'affichage de la maquette révèle, sous la forme de

liens d'interconnexion, la logique fonctionnelle sous-jacente. La figure 7.18 montre ces connexions sur la partie finale de la maquette.

Ces connexions ne sont pas une simple décoration : on l'a deviné, la maquette, si elle constitue une partition (une représentation dans le temps), constitue en même temps un programme dans le langage visuel OM. On peut donc l'évaluer en n'importe quel point, ou encore globalement. Dans ce cas, l'ensemble de la pièce est calculé. Les connexions sont des connexions paramétriques fonctionnelles, établies entre les entrées et les sorties des blocs, qui ne sont autres que des lambda-abstractions, au même titre que les boîtes (invocations) de patch insérées à l'intérieur d'autres patches. Les blocs de maquette sont donc des *patches temporels*, qui héritent de la classe `ompatch`. En tant que boîtes d'invocation apparaissant dans l'éditeur de maquette, ce sont des *boîtes temporelles*, qui héritent de `ombox`, et comportent des champs spécifiques décrivant leur position et extension dans le temps (figure 7.19).

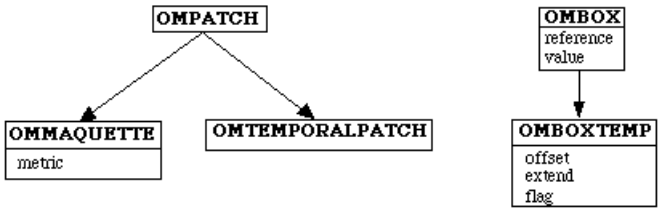


Figure 7.19. *Hiérarchie de classes pour l'implémentation des maquettes*

Lorsque l'on ouvre ces boîtes temporelles, on accède au patch temporel sous-jacent. Dans la figure 7.20, on a ouvert le dernier bloc à droite de Herma.

Du point de vue visuel, un patch temporel ne se différencie d'un patch ordinaire que par la présence de deux objets spéciaux : la boîte `self` (en haut à gauche) et la boîte `tempout` (en bas à gauche). Self `offre` une représentation de la boîte temporelle à l'intérieur même du patch. Ses *outlets* sont donc les champs de `omBoxTemp`, notamment `offset` et `extend`. D'autres informations, comme la dimension verticale du rectangle, sa position verticale, une image associée, etc. sont disponibles sur les autres *outlets*. Toutes ces informations sont des sources de valeurs utilisables dans l'algorithme de génération exprimé dans le patch. Ainsi, le sous-patch `genTimeIntv` prend-il en entrée le champ `extend` du bloc (sa durée) ainsi qu'un paramètre de densité, et génère-il un ensemble de positions et d'intervalles temporels pour la *factory* de type `chord-seq` située en bas du patch. En haut à droite, deux *AbsIn* (entrées d'abstraction) permettent de récupérer les données calculées par d'autres blocs et de les incorporer dans le calcul. Il s'agit ici de calculer le réservoir des hauteurs utilisé dans ce bloc, par union ensembliste des

deux réservoirs présents aux entrées. Les différentes abstractions effectuant le calcul stochastique des durées et la distribution des hauteurs et des intensités (genTimeintv, ArcSin, Gauss) ne sont pas détaillées. Tous les blocs de cette maquette sont bâtis sur le même modèle, les différences portant sur les paramètres comme la densité, le nombre d'entrées, et l'opération ensembliste appliquée aux entrées. La donnée de toutes ces caractéristiques, ajoutée à la position et la durée des blocs dans le temps, constitue véritablement la spécification de Herma par Xenakis.

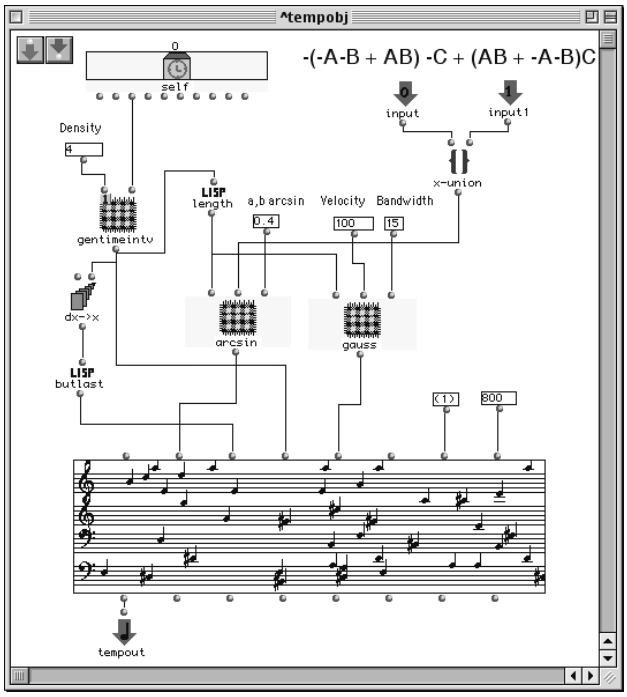


Figure 7.20. *Un patch temporel dans Herma*

La maquette constituant une partition aussi bien qu'un programme, les différents modes de visualisation mettent plus ou moins en avant chacun de ces deux aspects. La figure 7.21 montre un détail de la maquette affichée en notation musicale. Les blocs peuvent être alors ouverts, rendant accessible l'éditeur de la classe concernée (ici, la classe `chord-seq`, représentant une séquence d'accords disposés dans le temps).

La structuration hiérarchique peut être opérée par inclusion de maquettes à l'intérieur d'autres maquettes. L'exemple figure 7.22 montre une maquette dans laquelle ont été inclues, par glisser-déposer, deux instances de la maquette Herma décrite précédemment. Ces deux sous-maquettes ont été de surcroît compressées ou

étirées dans le temps, et assorties d'objets musicaux externes, des fichiers audio (en bas à gauche) et un fichier midi (en haut à droite).

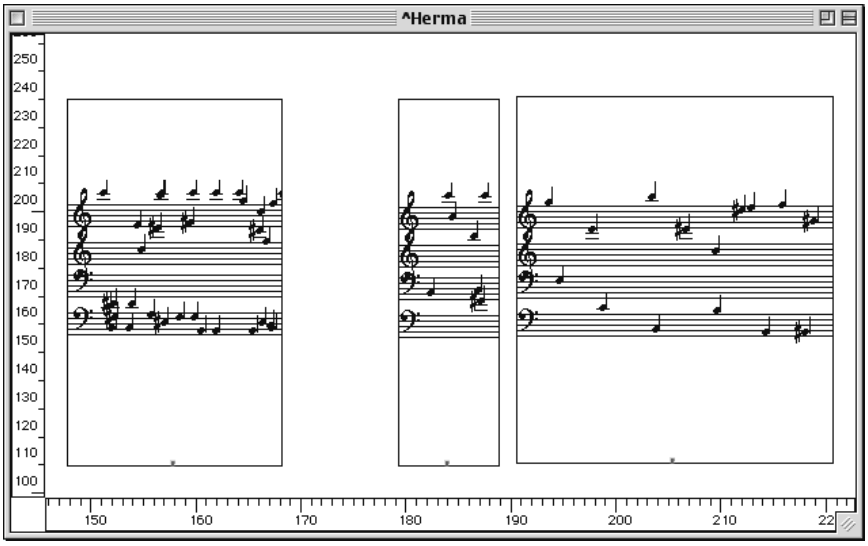


Figure 7.21. *Herma en mode notation, agrandissement d'un détail*

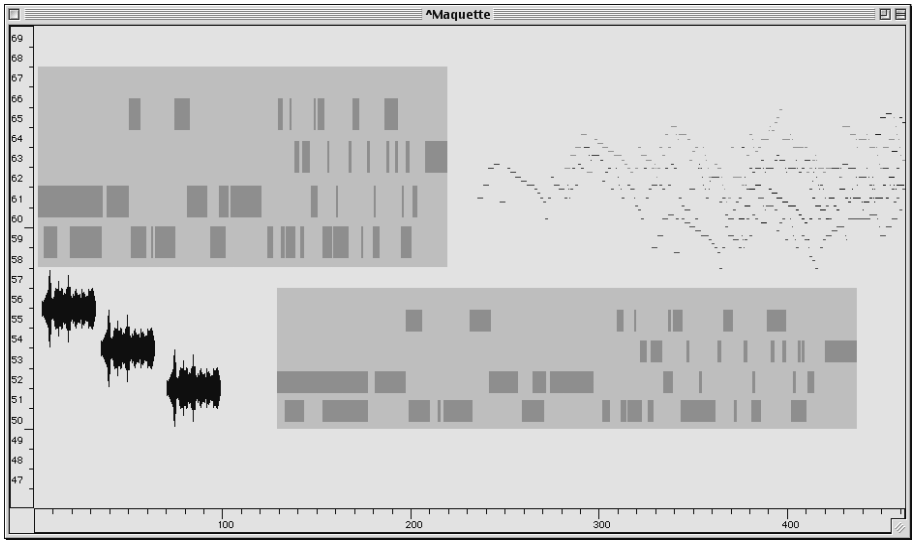


Figure 7.22. *Hiérarchie de maquettes*

Les aspects de logique temporelle sont réalisés à l'aide de marqueurs attachés à une position temporelle sur la règle horizontale. Ces marqueurs peuvent être attachés à la position de début ou de fin d'un bloc qui se trouve alors contraint. Une partie des relations d'Allen [ALL 83] est ainsi implémentée. La relation *meet*, par exemple correspond à un marqueur contraignant à la fois la fin d'un bloc et le début d'un autre. Lorsque l'utilisateur tente de déplacer un des deux blocs, ou de les étirer, ou de déplacer le marqueur, la relation se trouve toujours conservée (figure 7.23).

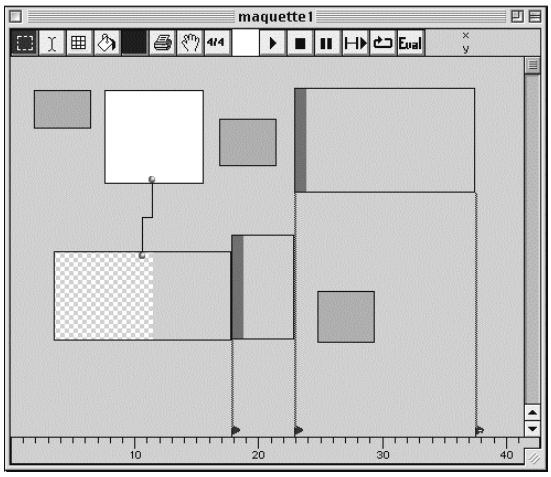


Figure 7.23. *Relations de logique temporelle*

La dualité partition/programme inhérente aux maquettes peut aussi s'exprimer selon la dualité relation de causalité/relation temporelle. La causalité entre deux blocs sera définie par rapport au « temps d'évaluation ». L'évaluation d'une maquette suit le même paradigme que pour les patches, à savoir que les blocs liés par des connexions forment un treillis orienté, correspondant à un ordre partiel (les boucles étant interdites). La propagation de l'évaluation va alors des éléments minimaux vers les éléments maximaux. D'autre part, les blocs entretiennent des relations de précédence selon l'ordre total du temps physique. Afin de mieux comprendre la façon dont une maquette peut représenter simultanément ces deux ordres, prenons l'exemple suivant (figure 7.24).

Cette dualité, qui est une des caractéristiques originales d'OM, a des conséquences importantes pour l'utilisation compositionnelle. Elle permet en effet d'envisager des situations dans lesquelles l'information « vient du futur », situations tout à fait familières pour le compositeur, mais en général irréalisables dans les environnements de composition ou de performance musicales, qui se réfèrent à un seul ordre chronologique.

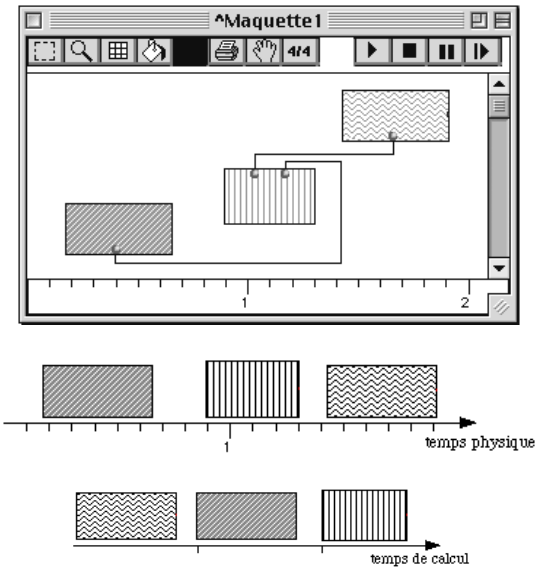


Figure 7.24. *Ordre causal et ordre temporel dans une maquette*

Le musicien a ainsi accès à quatre niveaux interdépendants de l'organisation musicale :

– le niveau statique de la forme, à savoir l'organisation temporelle/spatiale/ hiérarchique des blocs et des sous-maquettes ;

– le niveau dynamique de la forme, à savoir les relations fonctionnelles qui lient les blocs entre eux ;

– le niveau syntaxique, c'est-à-dire les modalités algorithmiques selon lesquelles se construit le discours au sein de chaque bloc ;

– le niveau du matériau, à savoir les données d'entrée (par exemple les ensembles A, B et C dans Herma).

Ces quatre axes d'organisation sont autant de logiques musicales qui s'interpénètrent, l'innovation dans OM étant de matérialiser cette interpénétration et d'en donner un contrôle interactif au compositeur. Ce contrôle débouche sur des possibilités d'expérimentations combinatoires significatives :

– recombinaison au niveau de la forme : les blocs sont réarrangés, dilatés ou compressés dans le temps ;

– modification des relations fonctionnelles (les connections) : ce sont alors les trajets des matériaux au sein de la pièce qui sont modifiés, sans changer l'organisation globale des blocs dans le temps ;

– modification syntaxique, c'est-à-dire changement de l'algorithmique contenue dans les blocs. C'est la forme du discours qui change, en conservant l'allure formelle et l'organisation temporelle de haut niveau.

La forme est conservée dans ses aspects statiques et dynamiques, la structure syntaxique est conservée, mais la substance sonore est changée (par exemple, la couleur harmonique).

7.6. Quelques exemples musicaux

Cette section décrit deux exemples types de l'utilisation d'OM par les compositeurs.

7.6.1. *Approche fonctionnelle*

L'approche fonctionnelle est peut être celle qui a été la plus utilisée par les compositeurs. En effet, ce style de programmation donne une réponse à la nécessité de disposer d'objets multidimensionnels et d'appliquer sur eux des opérations qui les modifient ou produisent des nouveaux objets. Dans ce sens, OM peut se concevoir comme une boîte à outils servant à faire des opérations musicales. De grande importance pour le compositeur, est la classification des dites fonctions par rapport au type de connaissance musical qu'elles représentent. Nous trouvons dans OM des fonctions orientées intervalles ; des fonctions de traitement harmonique ; d'interpolation d'accords, de motifs et d'objets en général. OM a donné de la force à la démarche fonctionnelle en musique et l'on a très vite vu que ce paradigme pouvait être appliqué au domaine de la synthèse. Tel est le cas dans « L'Amour de loin », le premier opéra de la compositrice finlandaise Kaija Saariaho [BAN 03]. La figure 7.25 résume le processus de transformation que subissent les matériaux compositionnels dans cette pièce.

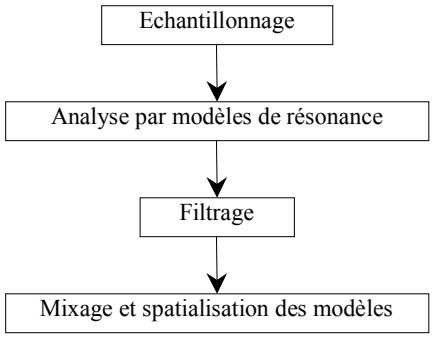


Figure 7.25. *Transformation des matériaux dans* L'Amour de loin

Le matériau sonore à la base est composé par des échantillons des voix parlées extraites du livret et des sons instrumentaux. Ces échantillons sont analysés en utilisant la technique de modèles de résonance. Le résultat de cette analyse est une liste de fréquences, amplitudes et largeurs de bande qui donnent beaucoup d'information sur le timbre des sons. Pour plus d'information sur les modèles de résonances [PBJ 86].

L'étape de filtrage consiste en le croisement d'une partie électronique issue d'une structure harmonique *a priori* avec un timbre issu de l'analyse par modèles de résonances. La partie électronique est d'abord écrite sous forme de partition. Les sons électroniques reposent sur des structures harmoniques qui sont propres à chaque personnage de l'opéra. La figure 7.26 présente les accords utilisés dans l'élaboration des sons électroniques pour le personnage Jaufré.

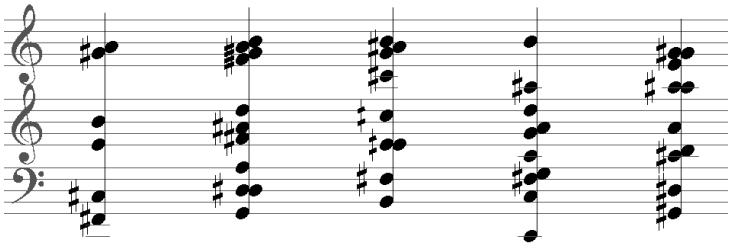


Figure 7.26. *Accords de Jaufré*

Le patch OpenMusic, figure 7.27, illustre l'extraction de modèle de résonance pour un échantillon de piano sur la note *fa*# 0.

Dans la figure 7.28, le premier accord du personnage Jaufré est utilisé. A chacune des fréquences des notes de l'accord sont associées, après un calcul d'interpolation spectrale, l'amplitude et la largeur de bande correspondantes du modèle instrumental. Ce patch OpenMusic effectue le filtrage.

Pour chacun des accords associés aux personnages, de nombreux filtrages ont été réalisés. Pour toute l'œuvre, environ trois cents filtres ont été générés. En considérant le nombre d'accords et d'échantillons instrumentaux analysés au regard des trois filtres utilisés lors de la synthèse, il est aisé de voir que le nombre de combinaisons des choix possibles est élevé.

Finalement, le mixage des sons réalisés précédemment est effectué en temps réel à l'aide du spatialisateur développé à l'Ircam.

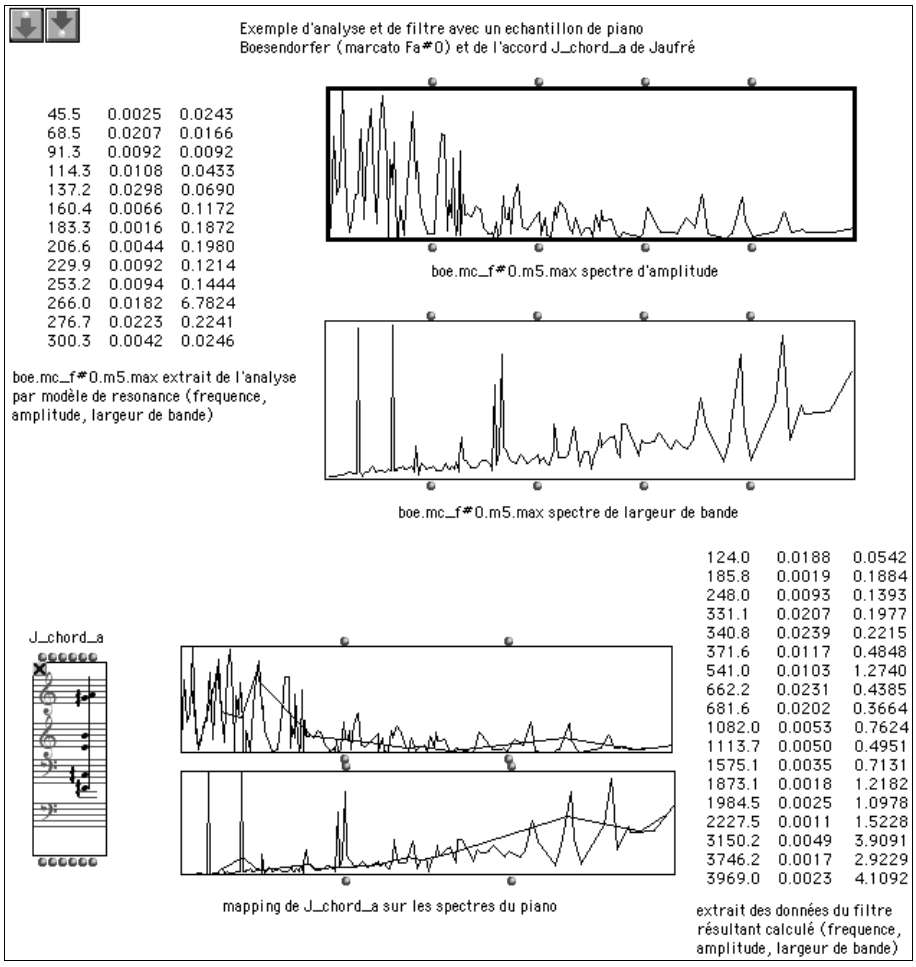


Figure 7.27. *Analyse par modèles de résonances d'un son de piano*

7.6.2. *Approche par Objets*

Un exemple d'utilisation approfondie des techniques par objets se trouve dans l'environnement OMChroma du compositeur Marco Stroppa. L'idée principale dans OMChroma consiste à affaiblir la dichotomie entre un son vu comme le résultat d'un processus de synthèse et comme une entité logique musicalement pertinente. *OpenMusic* fournit une bibliothèque d'objets musicaux, c'est-à-dire un ensemble de classes et de méthodes pour la représentation et le traitement de l'écriture musicale (c'est-à-dire, notes, accords, voix, polyphonies).

Au même titre, nous décrirons dans ce paragraphe, la bibliothèque d'objets de synthèse qui contient des entités de haut niveau pour le processus de synthèse de son. A la base des objets de synthèse, nous avons une représentation matricielle, où lignes et colonnes de la matrice sont associés aux paramètres de synthèse. Nous avons implémenté une classe abstraite appelée *array* avec un *champ* particulier *numcol* spécifiant le nombre de colonnes. La classe *array* n'est pas utilisable directement, elle doit être sous-classée en ajoutant des *slots* qui correspondront aux paramètres de synthèse et qui ajouteront des lignes à la matrice.

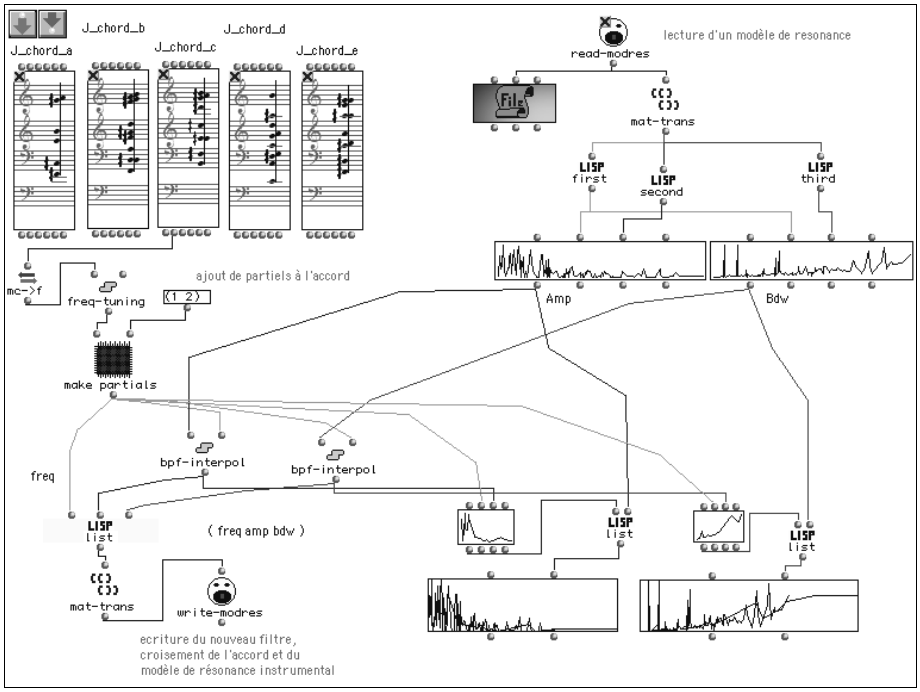


Figure 7.28. *Patch pour le filtrage*

Dans la figure 7.29, nous voyons la création d'une matrice à 20 colonnes et 5 lignes (*numcol* = 20 et 5 *slots* ajoutés). Si les valeurs des paramètres sont des nombres les lignes pourront être représentées sous forme de courbe, comme il est montré dans la figure 7.29. Une enveloppe sera clairement une matrice avec une seule ligne.

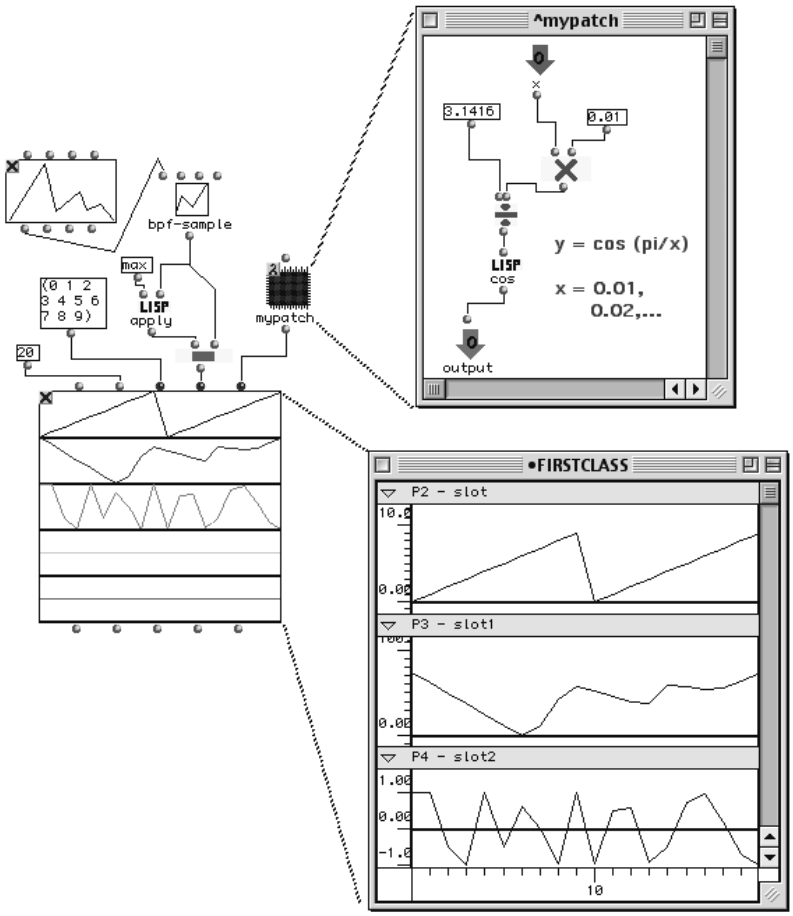


Figure 7.29. *Une instance de la classe array*

Il n'est pas nécessaire de remplir toutes les lignes, dans notre exemple, seulement trois des cinq slots ont été fournis. Pour les deux autres, on utilise les valeurs par défaut données lors de la définition de la classe, les courbes en question seront des constantes. Une particularité de la classe *array* est que les valeurs des *slots* peuvent être données sous diverses formes. Dans la figure 7.29, les trois entrées de paramètres se font de gauche à droite. Ce sont successivement :

– une liste d'entiers, laquelle est répétée cycliquement jusqu'à atteindre le nombre de colonnes ;

– le résultat d'un algorithme visuel (patch OpenMusic) ;

– une lambda expression visuelle (montrée dans la fenêtre *mypatch*, y = cos(Pi/x)).

Quand les valeurs des *slots* sont des fonctions, la matrice ne garde pas les valeurs de l'application de la fonction, mais la fonction elle-même. Les valeurs, produits de l'application de cette fonction, seront calculées lorsque cela sera nécessaire. Cette technique est appelée *évaluation paresseuse*.

Les *Arrays* peuvent être construits à partir d'objets musicaux. Pour cela, il faut définir un algorithme d'interprétation, comme montré dans la figure 7.30.

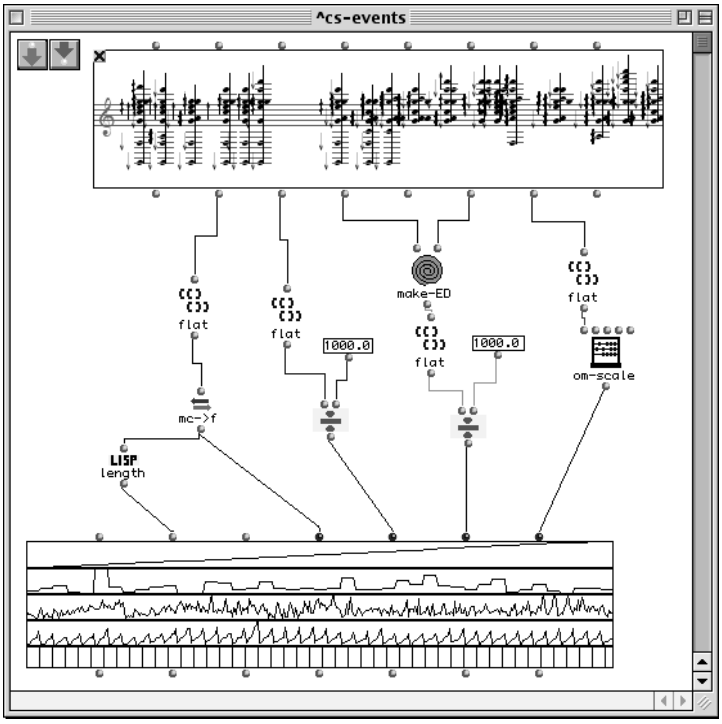


Figure 7.30. *Création d'un array à partir d'un objet musical*

Tout modèle de génération sonore a besoin de données de contrôle, qui dépendent du synthétiseur utilisé. Cependant, plusieurs paramètres de contrôle peuvent être conceptuellement les mêmes, indépendamment de leur implémentation. Un vibrato, par exemple, sera toujours utilisé comme un vibrato quel que soit la façon dont il est codé.

Les *arrays* peuvent être groupés en sous-classes abstraites qui, grâce au polymorphisme, peuvent être interprétés par un synthétiseur donné. Ainsi des données similaires n'auront pas besoin de structures différentes pour être traitées par

différents synthétiseurs. Les algorithmes de production de données sont ainsi isolés du moteur de synthèse. Cette étape d'interprétation relève de ce que nous appelons un « synthétiseur » virtuel. Le polymorphisme est concentré dans une fonction générique appelée *synthesize*. Cette fonction est spécialisée par le nom de synthétiseur désiré, elle prend une liste d'*arrays* en filtrant toutes les instances qui ne sont pas des sous-classes de la classe abstraite associée au synthétiseur. Cette fonction traduit les *arrays* dans un format approprié : fichiers *orc* et *score* pour *Csound* (un synthétiseur logiciel très populaire), un fichier *SDIF* pour *Chant* (synthétiseur développé à l'Ircam), paquets *OSC* (protocole de communication développé au CNMAT à Berkeley) pour *Max/msp* ou *jMax* (Environnements de synthèse temps-réel développés à l'Ircam et par la société Cycling 74), etc. L'adjonction de nouveaux synthétiseurs au système sera réduite à la définition de nouvelles méthodes pour *synthesize*. La figure 7.31 montre un exemple de la notion de synthétiseur virtuel. Le matériau de base vient d'une analyse d'un son de cymbale. Le côté gauche de la figure montre une synthèse utilisant *Chant*. Le côté droit renvoie les mêmes données d'analyse à *Csound*. Les résultats des deux synthèses sont presque identiques.

Cependant, il est peu probable qu'un compositeur pense un processus de synthèse seulement en fonction d'un son unique. D'après notre expérience, un compositeur verra ce processus plutôt comme un modèle qui génère une famille de sons pouvant aussi être étendue progressivement. De ce point de vue, la notion de son est remplacée par celle de potentiel sonore [STR 00]. Ce dernier a à voir avec l'idée d'une classe d'équivalence de tous les résultats possibles d'un modèle de synthèse. Ainsi, les deux sons de la figure 7.31 ne sont pas équivalents bien qu'il produisent le même résultat. Nous devons alors considérer le son non seulement comme un résultat, mais aussi en prenant en compte la façon (l'algorithme) dont il a été produit et ses possibles extensions. Notre but sera alors de fournir une représentation (informatique et graphique) qui rende compte en même temps du contrôle de la synthèse et de sa pertinence pour la composition.

7.7. Conclusion

OpenMusic est désormais enseigné dans plusieurs universités et conservatoires et utilisé par un nombre croissant de compositeurs et de musicologues.

L'utilisation extensive d'OpenMusic par des compositeurs de musique contemporaine a indéniablement eu pour effet de relativiser la complexité de certaines techniques en vogue depuis la révolution dodécaphonique et ses suites. Lorsqu'un ordinateur peut calculer très rapidement toutes les variantes combinatoires issues d'un formalisme donné, par exemple le formalisme sériel, la valeur musicologique ne peut plus venir simplement de ce principe formel, mais de son

adéquation réelle à la perception. Corrélativement, l'ordinateur permet d'explorer des champs expérimentaux d'une complexité réelle et qui étaient inaccessibles auparavant, comme par exemple le champ du timbre dans sa représentation spectrale.

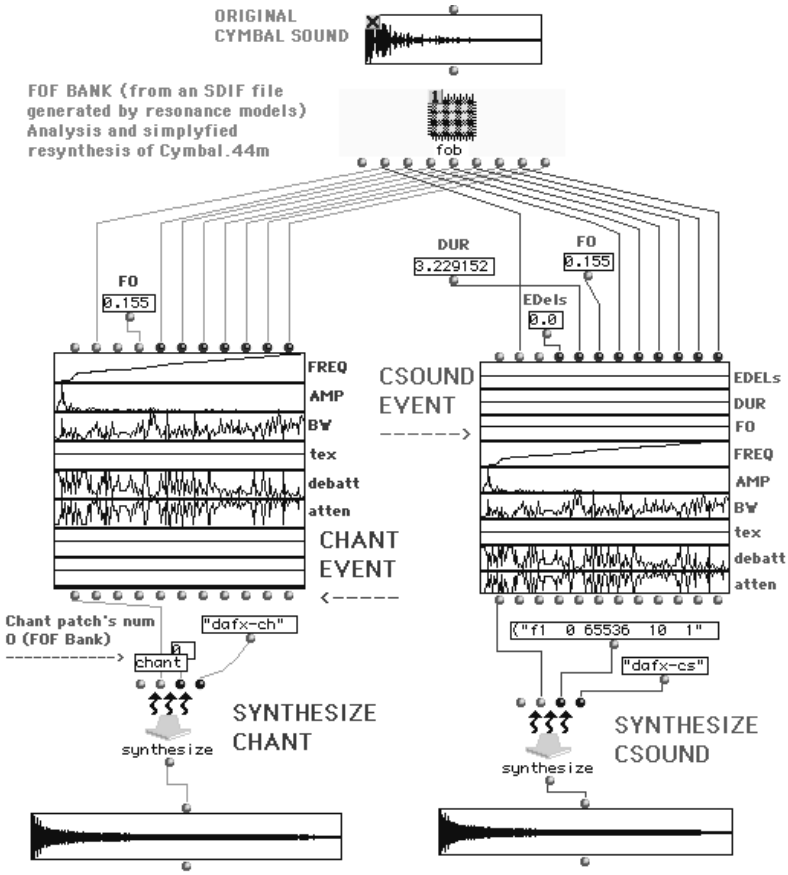


Figure 7.31. *Création d'un array à partir d'un objet musical*

Parallèlement, OpenMusic a favorisé la démarche expérimentale qui permet de soumettre un très grand nombre d'instances musicales issues d'un formalisme au test de la musicalité. Plus encore, elle autorise l'expérimentation sur les formalismes eux même qui peuvent être testés et en retour modifiés ou abandonnés s'ils ne remplissent pas leurs promesses. Aux notions d'unicité de l'œuvre et de son créateur, elle tend à substituer l'idée du modèle d'œuvre et de la coopération entre créateurs et scientifiques. C'est peut être là que réside son aspect le plus original, et donc le plus polémique.

Enfin, OpenMusic propose un renouvellement de l'idée de notation qui devient dynamique, incluant les mécanismes génétiques de l'œuvre. La partition constitue d'une certaine manière sa propre analyse, ce qui est déjà le germe d'un bouleversement des habitudes musicologiques. Nous avons essayé avec OpenMusic de construire un langage qui encapsule de manière consubstantielle la notion de notation, à savoir notation du résultat (une partition musicale) mais aussi notation du processus aboutissant à ce résultat (programme visuel).

7.8. Bibliographie

[ABC 95] ABADI M., CARDELLI L., « A Theory of Primitive Objects », *Digital Equipment Corporation, Systems Research Center*, 1995.

[AGO 98] AGON C., OpenMusic: Un langage visuel pour la composition musicale assistée par ordinateur, Thèse de doctorat en informatique, Université Paris VI., 1998.

[AAD 98] AGON C., ASSAYAG G., DELERUE O., RUEDA C., « Objects, Time and Constraints in OpenMusic », *Proceedings of ICMC.'98*, San Francisco, 1998.

[ALL 83] ALLEN J.F., « Maintaining Knowledge about Temporal Intervals », *Communications of the ACM*, 16,11, 832- 843, 1983.

[ASS 85] ASSAYAG G., CASTELLENGO M., MALHERBE C., « Functional integration of complex instrumental sounds in music writing », *Proceedings of ICMC'85*, Vancouver, 1985.

[ASS 86] AMIOT E., ASSAYAG G., MALHERBE C., RIOTTE A., « Duration structure generation and recognition in music writing », *Proceedings of ICMC'86*, La Haye, 1986.

[ASS 99] ASSAYAG G., RUEDA C., LAURSON M., AGON C., DELERUE O., « Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic », *Computer Music Journal*, vol. 23, n° 3, décembre 1999.

[BAL 92] BALABAN M., EBCIOGLU K., LASKE O., *Understanding Music with AI: Perspectives on Music Cognition*, MIT Press, Cambridge, 1992.

[BAN 03] BATTIER M., NOUNO G., « L'électronique dans l'opéra de Kaija Saariaho, L'Amour de loin », *Musurgia*, 2003.

[CAW 85] CARDELLI L., WEGNER P., « On Understanding types, data Abstraction and polymorphism », *Computing Surveys*, vol. 17, n° 4, 1985.

[CAS 97] CASTAGNA G., *Object-oriented programming: a Unified Foundation*, Birkhäuser, Boston, 1997.

[CAS 98] CASTAGNA G., « Foundations of Object-oriented programming », *Tutorial notes, Workshop ETAPS'98*, Lisbonne, 1998.

[CHA 87] CHANG S.K., « Visual Languages: A Tutorial and Survey », *IEEE Software Magazine*, vol. 4, n° 1, 29-39, janvier 1987.

[CHA 97] CHANG S.K., POLESE G., OREFICE S., TUCCI M., A Methodologie and Interactive Environment for Iconic Language Design, Thèse de doctorat, Université de Pittsburgh, 1997.

[COU 90] COURTOT F., CARLA: Knowledge acquisition and induction for computer assited composition, Rapport de recherche, Ircam, Paris, 1990.

[DEH 94] DESAIN P., HONING H., CLOSe to the edge? Advanced object oriented techniques in the representation of musical knowledge, Rapport de recherche CT-94-13, Institute for Logic, Language and Computation, Amsterdam, 1994.

[HAA 95] VOLKER H., « Formal Semantics of Visual languages Using Spatial Reasoning » *Proceedings, 11th Symposium on Visual Languages*, Darmstadt, 1995.

[HAA 96] VOLKER H., « A Fully Theory for Describing Visual Notations », *Proceedings of the International Workshop on the Theory of Visual Languages*, Gubbio, Italie, 1996.

[JBK 70] DAHL O.J., MYHRHAUG B., NYGAARD K., SIMULA – Common Base Language Rapport technique NS-22, Norwegian computing center, Oslo, 1970.

[LAU 96] LAURSON M., « PATCHWORK: A Visual Programming Language and some Musical Applications », *Sibelius Academy*, *Studia Musica*, n° 6, Helsinki, 1996.

[MAM 96a] MARRIOTT K., MEYER B., « Towards a Hierarchy of Visual Languages » *Proceedings of the 12th International IEEE Workshop on Visual Languages*, Université Monash, 1996.

[MAM 96b] MARRIOTT K., MEYER B., « Formal Classification of Visual Languages » *Proceedings of the Ist International Workshop on Theory of Visual Languages*, Gubbio, Italie, 1996.

[PAC 94] PACHET F., « The MusES system: an environment for experimenting with knowledge representation techniques in tonal harmony », *First Brazilian Symposium on Computer Music, SBC&M'94*, Caxambu, Minas Gerais, Brézil, 3-4 août 1994.

[PAE 93] PAEPCKE A., *Object-Oriented Programming – The CLOS Perspective*, MIT Press, Cambridge, 1993.

[PBJ 86] POTARD Y., BAISNÉE P.F., BARRIÈRE J.B., « Experimenting with Models of Resonance Produced by a New Technique for the Analysis of Impulsive Sounds », *Proceedings of ICMC*, La Haye, Berkeley, 1986.

[POP 91] POPE S., Introduction to MODE: The Musical Object Development Environment, MIT press, Boston, 1991.

[ROC 85] RODET X., COINTE P., FORMES Composition et ordonnancement de processus, Rapport de recherche n° 36, Ircam, Paris, 1985.

[SAI 93] SAINT-JAMES E., *La programmation applicative (de LISP à la machine en passant par le lambda-calcul)*, Hermès, Paris, 1993.

[STE 98] STEELE G., *Common LISP The language, 2e edition*, Digital Press, 1998.

[STR 00] STROPPA M. « Paradigms for the high-level musical control of digital signal processing », *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00)*, Verona, Italie, 2000.

[TAU 91] TAUBE H., « Common music: A music composition language in Common Lisp and CLOS », *Computer Music Journal, 15(2)*, 1991.

[XEN 91] XENAKIS I., *Formalized Music, Thoughts and Mathematics in Composition,* Pendragon Press, Stuyvesant, 1991.

Chapitre 8

Modèles temporels pour la synthèse musicale

8.1. Introduction

Nous présentons, dans ce chapitre, des formalismes utiles dans le cadre de l'aide à la composition de musique écrite. Nous restreignons notre champ d'investigation aux représentations de l'écriture musicale qui sont en adéquation avec des situations fréquentes en phase de création. Ces formalismes sont donc adaptés aux problèmes d'édition de pièces de musique (modifier, changer des paramètres sonores ou musicaux, enlever ou ajouter des objets sonores ou des parties musicales), harmoniser, produire des arrangements, transcrire, etc.

L'aide à la création consiste à bâtir, à partir du vide, une description de plus en plus précise d'un projet. Mais la composition musicale ne consiste pas toujours à écrire de gauche à droite. Des parties peuvent être produites séparément et assemblées ultérieurement selon des relations temporelles imposées par le compositeur. En conséquence, une édition peut intervenir n'importe où dans la pièce et n'importe quand. L'imprécision tient donc une grande place dans ce processus de l'aide à la création, car il consiste à préciser de plus en plus un projet. Cela se concrétise par une incertitude sur certaines valeurs et sur certaines relations temporelles entre les composantes musicales.

En ce qui concerne le temps, les dates absolues n'ont généralement pas d'importance. Seule la connaissance des relations temporelles relatives importe. La composition s'effectue aussi sur plusieurs échelles de temps, la précision en temps pouvant varier de la milliseconde à la minute.

Chapitre rédigé par Myriam DESAINTE-CATHERINE.

Les critères de qualité d'un bon système de représentation temporelle peuvent se mesurer à l'efficacité des opérations d'édition de base, à l'expressivité du modèle temporel, à la part d'incertitude qu'il peut accepter et à la facilité de changement d'échelle de temps.

Dans ce chapitre, nous synthétisons et comparons les trois types de représentations classiques suivantes :

1) les systèmes de pistes avec dates absolues sont utilisés dans les éditeurs de partitions et les séquenceurs de sons ou d'événements MIDI. Nous verrons que ces systèmes sont bien adaptés à la transcription musicale. Ils sont très répandus et communément utilisés pour la création. Cependant, ils ne sont pas flexibles, car ils ne proposent qu'une seule sorte d'organisation temporelle qui est la superposition de pistes (avec quelques extensions) ;

2) les systèmes hiérarchiques ont été introduits afin d'accroître l'efficacité de l'édition, tout en restant proches de la représentation musicale que le compositeur a en tête. Ils sont basés sur des opérateurs temporels. La puissance d'expression du système dépend des opérateurs choisis et du contexte de mise en œuvre. Nous verrons, par exemple, que le chevauchement n'est pas toujours facilement représenté. Par contre, une certaine dose d'incertitude peut intervenir en ce qui concerne les durées et les dates absolues des données ;

3) dans les systèmes de relations temporelles, la musique est décrite par un ensemble de propriétés explicites. Ce genre de système est utile pour le raisonnement. Par contre, l'accès aux données est rendu difficile en l'absence de ligne de temps et d'origine.

La mise en œuvre des systèmes du second type varie selon les contextes. Le contexte fonctionnel est particulièrement développé dans ce chapitre, car il est simple et puissant. De même, le contexte grammatical, qui est aussi basé sur des formalismes classiques bien établis, est présenté dans le détail. Nous fournissons aussi quelques programmes en scheme pour mieux faire comprendre notre propos. Mais nous n'abordons pas la représentation orientée objets de ces systèmes. Le lecteur intéressé pourra consulter une étude de Mira Balaban sur ce sujet [BAL 93].

Dans ce chapitre, nous parcourons ces différents systèmes et leurs formalismes associés au moyen d'exemples afin de mettre en évidence leurs intérêts et leurs faiblesses. Nous commençons par comparer les avantages de la représentation hiérarchique par rapport à ceux de la représentation à base de pistes. Puis, nous introduisons les relations temporelles dans le cadre logique, car elles offrent des possibilités nouvelles par rapport aux opérateurs. Nous introduisons ensuite les systèmes basés sur les grammaires attribuées qui permettent des évaluations plus générales que dans le cadre fonctionnel.

Le lecteur sera peut-être un peu désappointé de constater que tous ces systèmes ont leurs avantages et leurs défauts et qu'aucun d'entre eux n'est totalement parfait. Mais nous espérons que ce constat stimulera certains lecteurs et motivera de nouvelles recherches sur le sujet, en combinant de façon adéquate certains aspects de ces systèmes ou en cherchant de nouveaux formalismes mieux adaptés au contexte musical. C'est dans ce but que nous présentons ici certains problèmes ouverts.

8.2. Les systèmes à base de pistes

Dans un système à base de pistes, une pièce de musique est représentée par un ensemble de parties instrumentales. Chaque partie instrumentale est elle-même un ensemble d'objets sonores, c'est-à-dire des sons ou des notes. L'étude que nous présentons dans cette section va montrer les limites de ces systèmes élémentaires de représentation : seul l'ajout de notes à la fin du texte est optimal. Les autres opérations offrent des choix limités dans leur implémentation par rapport aux besoins de la création ou par rapport aux lois musicales classiques. L'incertitude ne peut pas être représentée. De plus, les objets manipulés sont des atomes (notes ou sons), aucune structure musicale ne pouvant être définie dans ce système.

8.2.1. *Un extrait d'une partie instrumentale*

Considérons, comme premier exemple, le motif numéro 1 qui est constitué d'une seule partie (voir figure 8.1).

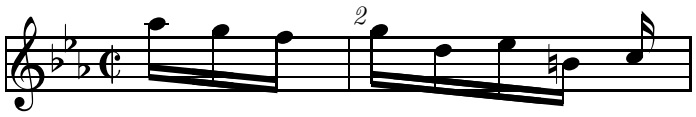


Figure 8.1. *Motif musical numéro 1*

Les atomes de cet extrait sont des notes. Chaque note peut se représenter par un quadruplet $< s, p, v, d >$ où :

– s représente la position temporelle absolue de la note par rapport au début de la pièce. Conformément à l'écriture, nous utiliserons la noire comme unité ;

– p représente la hauteur de la note, notée par son nom et son octave ;

– v représente le volume de la note. Soit $\{pp, p, mp, mf, f, ff\}$ l'ensemble des valeurs symboliques de volume de jeu ;

– d représente la durée de la note dans la même unité que la position temporelle.

Nous laissons de côté les informations écrites concernant la clef, la tonalité et la mesure pour nous intéresser plus précisément aux positions et aux durées des notes. Le motif musical peut se représenter par l'ensemble des notes suivant :

$$\{<0, a\flat 5, p, \tfrac{1}{4}>, <\tfrac{1}{4}, g5, p, \tfrac{1}{4}>, <\tfrac{1}{2}, f5, p, \tfrac{1}{4}>, <\tfrac{3}{4}, g5, p, \tfrac{1}{4}>,$$
$$<1, d5, p, \tfrac{1}{4}>, <\tfrac{5}{4}, e\flat 5, p, \tfrac{1}{4}>, <\tfrac{3}{2}, b\natural 4, p, \tfrac{1}{4}>, <\tfrac{7}{4}, c5, p, \tfrac{1}{4}>\}$$

Dans cet ensemble, les notes sont totalement indépendantes les unes par rapport aux autres. En effet, toute modification, toute suppression ou tout ajout d'une note n'a aucune influence sur les autres notes. Par exemple, si le compositeur supprime la troisième note de notre motif et que cette opération est implémentée par la simple suppression du quadruplet lui correspondant dans l'ensemble ci-dessus, la note sera implicitement remplacée par un silence et donnera lieu au motif représenté sur la figure 8.2. Ce comportement peut être satisfaisant dans certains cas. Mais parfois, le compositeur peut souhaiter un décalage des notes suivantes de la séquence de façon à combler le silence. La représentation ensembliste n'est pas bien adaptée à l'implémentation de ce mécanisme. En effet, pour implémenter la suppression avec décalage, il faut parcourir potentiellement tout l'ensemble à la recherche de toutes les notes comportant une valeur de *s* supérieure à celle de la note supprimée afin de les décaler, ce qui peut prendre du temps dès que la pièce de musique atteint une certaine taille.

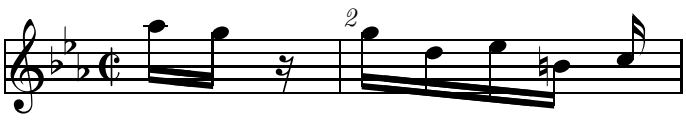


Figure 8.2. *Motif musical numéro 1 après la suppression de la troisième note*

Supposons maintenant que le compositeur change la quatrième note du motif 1 en une noire. Si cette opération est implémentée par une simple modification de la valeur *d* de cette note, un accord avec les trois notes suivantes va apparaître dans le résultat. Celui-ci peut s'écrire comme indiqué sur la figure 8.3. Là encore, le compositeur peut souhaiter un autre comportement qui consisterait à conserver la structure séquentielle des notes. L'implémentation de la modification d'une valeur de *d* avec décalage consiste alors aussi à rechercher toutes les notes comportant une valeur de *s* supérieure à celle de la note modifiée et à décaler toutes leurs valeurs.

Pour optimiser ces recherches, il faut représenter l'ordre temporel des notes selon leur valeur de *s*. On peut utiliser un tableau ordonné en lignes selon les valeurs de *s* croissantes et comportant en colonne les notes admettant la même valeur *s* (voir tableau 8.1) ou un arbre d'intervalle [COR 94]. Par contre, la mise à jour de toutes ces valeurs de *s* reste toujours à faire et peut s'avérer lourde dès que la pièce atteint

une taille importante. Nous verrons dans la section suivante qu'une solution à ce problème est l'utilisation de dates relatives dans une représentation hiérarchique de la pièce.

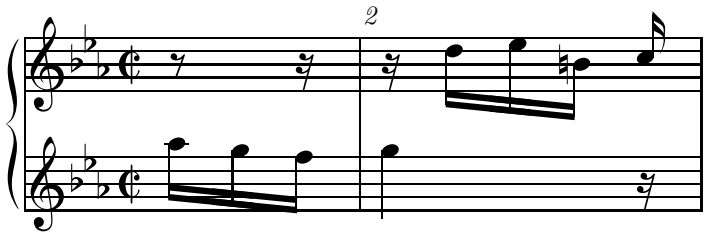


Figure 8.3. *Motif musical numéro 1 après la transformation de la quatrième double-croche en une noire*

Index	1	2	3	4	5	6	7	8
Position temporelle	0	1/4	1/2	3/4	1	5/4	3/2	7/4
Note	$a\flat 5$	$g5$	$f5$	$g5$	$d5$	$e\flat 5$	$b\natural 4$	$c5$
Volume	p	p	p	p	p	p	p	p

Tableau 8.1. *Tableau d'accès aux notes du motif 1*

8.2.2. *Plusieurs parties instrumentales*

Généralement, une pièce musicale comporte plusieurs parties instrumentales. L'organisation de leurs notes est régie par des lois musicales classiques (contrepoint ou harmonie) ou des contraintes imaginées par le compositeur.

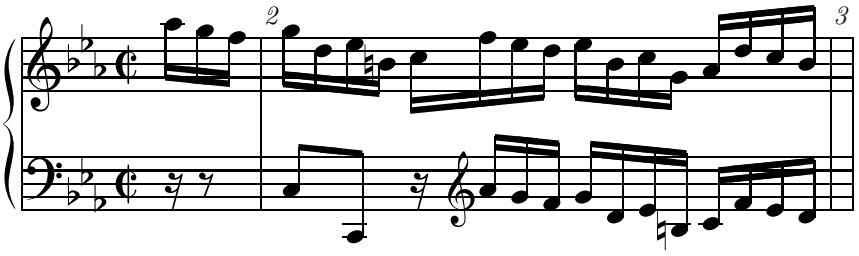


Figure 8.4. *Les deux premières mesures de l'Allemande de J.S. Bach*

Considérons un extrait de la Partita de J.S. Bach (voir figure 8.4) et envisageons une édition consistant à supprimer la cinquième note. Dans le système que nous explorons ici, nous nous restreignons à n'envisager que deux sortes de relations entre les parties de la main droite et de la main gauche : soit elles sont totalement indépendantes (ce qui est très rare, surtout en musique classique), soit leurs notes doivent conserver leurs organisations verticales relatives les unes par rapport aux autres.

Dans le premier cas, la suppression s'effectue sans décalage sur l'autre partie (voir figure 8.5). On constate que les relations harmoniques sont modifiées : une harmonie en *lab* majeur remplace l'harmonie initiale en *fa* mineur à l'endroit de la première note décalée. De plus, certains mouvements harmoniques sont inversés. Par exemple, le mouvement direct des accords correspondant aux neuvièmes et dixièmes notes à la main droite du motif initial se transforme en mouvement contraire.

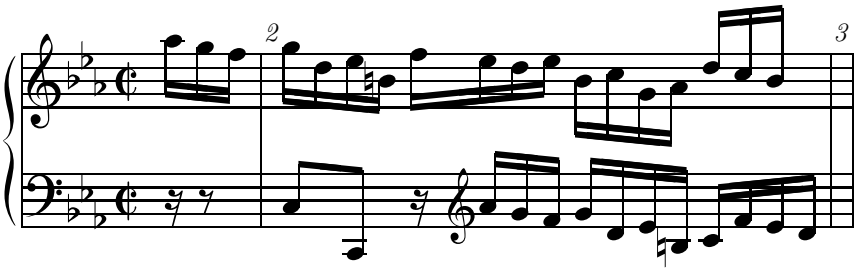


Figure 8.5. *Résultat de la suppression de la cinquième note de la deuxième mesure de la partie de la main droite avec décalage physique des notes suivantes de la main droite et sans décalage à la main gauche : l'organisation verticale est modifiée*

Dans le second cas, le décalage s'effectuant parallèlement sur les deux parties, l'harmonie et les mouvements sont bien conservés (voir figure 8.6). Mais un accord serait apparu s'il y avait eu une note à la place du silence à la main gauche.

8.2.3. *Utilité pratique*

Les exemples de manipulation précédents ont montré que cette représentation est bien adaptée à l'écriture de gauche à droite. Tout ajout à la fin est traité de façon optimale. Par contre, toute insertion interne ou toute modification de durée d'un atome interne implique une perturbation de la cohérence de l'écriture. Ce système de représentation ne permet que deux choix arbitraires de traitement de l'incohérence par partie instrumentale.

Dans le dernier exemple présenté, il semble évidemment préférable d'adopter le deuxième mécanisme, car nous avons affaire à un extrait d'une pièce déjà composée.

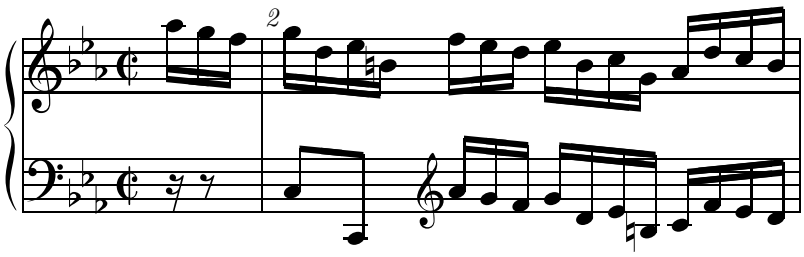


Figure 8.6. *Résultat de la suppression de la cinquième note de la deuxième mesure de la partie de la main droite avec décalage physique des notes suivantes de la main droite et avec décalage physique à la main gauche : l'organisation verticale main droite et gauche est conservée, mais l'organisation de la main gauche est modifiée.*

Mais dans un contexte de création, le compositeur peut transformer son écrit tout en souhaitant conserver certaines des relations précédentes et en modifier d'autres. La simple représentation ensembliste ne peut faire face à cette complexité.

Le cadre d'utilisation de cette représentation est donc plutôt celui de la transcription qui nécessite moins de mises à jour des parties déjà écrites que dans le cas de la composition musicale.

8.3. Les systèmes hiérarchiques

Afin de pallier l'inefficacité des mises à jour des systèmes à base de pistes, des systèmes de valeurs temporelles relatives ont été introduits. Ces systèmes utilisent une structuration arborescente des données permettant l'utilisation de valeurs temporelles relatives par rapport à chaque niveau. La part d'incertitude concernant les valeurs temporelles et les relations entre les parties varie selon les systèmes.

Nous utilisons ici un formalisme basé sur celui de Mira Balaban [BAL 89a, BAL 89b, BAL 91, BAL 98] pour décrire la struturation hiérarchique.

8.3.1. *Combinaison temporelle*

Une pièce musicale est considérée comme une combinaison le long d'un axe temporel de sous-parties indépendantes. L'exemple du tableau 8.2 illustre cette notion. Ce tableau décrit une vue structurée d'une pièce p constituée de trois parties :

1) la première partie notée p_1 commence au point temporel de valeur -30,

2) la deuxième partie notée p_2 commence au point temporel de valeur 0,

3) la troisième partie notée p_3 commence au point temporel de valeur 50.

Pièce composée p		**Sous-partie p_1**	**Sous-partie p_2**	**Sous-partie p_3**
−30	start	start		
−20		time-stamp		
−5			time-stamp	
0			start	
25		clip		
50			clip	start/time-stamp
80	clip			clip

Tableau 8.2. *Combinaison temporelle*

Chaque sous-partie a son propre axe temporel avec son origine associée à une date d'occurrence représentée par l'attribut *time-stamp*, exprimé dans le repère temporel de la partie la contenant et ses points de début (*start*) et de fin (*clip*). Une pièce ne commence pas nécessairement en son origine. Elle peut admettre une introduction, impliquant un point de début antérieur à l'origine (comme p_1) ou un délai impliquant un point de début postérieur à l'origine (comme p_2).

Cette notion de hiérarchie temporelle permet d'avoir une vue unifiée des pièces et de leurs composantes, chacune d'elles pouvant être aussi une hiérarchie temporelle. En effet, une pièce musicale est définie récursivement de la façon suivante. Une pièce de musique structurée est soit :

1) une pièce de musique structurée élémentaire, c'est-à-dire une occurrence d'un son ;

2) une pièce de musique structurée composée, c'est-à-dire une collection de pièces musicales associées à leur date d'occurrence.

8.3.2. *Exemples*

Un canon à deux voix (voir tableau 8.3) est une combinaison de deux sous-parties identiques dont les échelles temporelles sont décalées d'une durée d.

Un accord à trois notes de durée d (voir tableau 8.4) est une combinaison simultanée d'objets élémentaires.

8.3.3. *Edition hiérarchique*

Nous illustrons ici l'intérêt des dates relatives dans le cadre d'une édition. Supposons que l'utilisateur souhaite supprimer la partie p_2 de la pièce p décrite au paragraphe 8.3.1 et la remplacer par p_3. Contrairement aux systèmes à base de pistes,

p_0		p	p
0	start	start/time-stamp	
d			start/time-stamp
		clip	
	clip		clip

Tableau 8.3. *Un canon à deux voix*

p_0		n_1	n_1	n_3
0	start	start/time-stamp	start/time-stamp	start/time-stamp
d	clip	clip	clip	clip

Tableau 8.4. *Un accord à trois notes*

cette opération peut être implémentée très efficacement en temps, car seule la date d'occurrence de p_3 est modifiée. En effet, les dates des sous-parties de p_3 sont inchangées puisqu'elles sont définies relativement à celle de p_3.

8.4. Les systèmes fonctionnels hiérarchiques

Dans les systèmes fonctionnels, la hiérarchie correspond au terme induit par composition d'applications d'opérateurs structurels sur des atomes musicaux ou sur d'autres termes. Dans un projet de recherche et de développement [BAL 89a] concernant la représentation de données musicales et le raisonnement temporel, qui a commencé au milieu des années 1980, Mira Balaban introduit les structures musicales (*musical structures*). Les structures musicales sont définies à partir d'un opérateur temporel principal appelé *concaténation musicale*. Le terme induit par l'application de cet opérateur, ainsi que d'autres opérateurs de divers types (structurels, temporels, musicaux, etc.), représente à la fois la pièce et sa genèse. Les valeurs des positions temporelles et des durées sont situées sur les feuilles du terme. A la fin de cette section, nous illustrons cette approche au moyen des opérateurs rythmiques définis dans OpenMusic (développé à l'IRCAM).

8.4.1. *Représentation des structures musicales*

Les atomes musicaux de cette représentation sont définis sur la notion de *son sans durée*. Un son sans durée est une description sonore excluant la durée. Par exemple, une note MIDI est décrite par deux événements constitués de valeurs de volume, de hauteur et de canal. Sa durée est implicite et indépendante de ses autres caractéristiques.

Une pièce de musique structurée élémentaire est une paire $[p, d]$ associant un terme p représentant un objet sonore avec un terme d représentant un nombre non négatif. Le terme d est appelé la durée de $[p, d]$. Le terme p associe une hauteur avec un volume. Par exemple, les notes et les soupirs peuvent se représenter par des pièces structurées élémentaires. Ainsi, les notes et le soupir de la portée représentée sur la figure 8.7 peuvent se coder de la façon suivante, en utilisant les mêmes notations qu'au paragraphe 8.2.1 : $p_1 = [(g4, p), 1]$, $p_2 = [REST, 1]$, $p_3 = [(c4, p), 4]$.

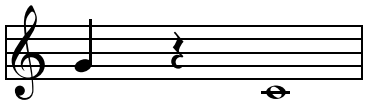


Figure 8.7. *Exemple de portée*

8.4.1.1. *Notation*

Une pièce de musique structurée composée peut être décrite au moyen d'une notation de listes basée sur l'opérateur « . » de concaténation musicale :

– nil décrit la pièce de musique structurée composée vide ;

– $(p_1 \ @ \ t \ . \ p_2)$ pour $p_1 \neq nil$ décrit la pièce représentée par p_2 augmentée de p_1 dont la date d'occurrence *time-stamp* est t.

Pour simplifier la notation, la structure musicale :

$$\big(p_1 \ @ \ t_1 \ . \ \big(p_2 \ @ \ t_2 \ . \ ... \ . \ (p_n \ @ \ t_n \ . \ tail)...\big)\big)$$

est notée :

$$(p_1 \ @ \ t_1 \ p_2 \ @ \ t_2 \ ... \ p_n \ @ \ t_n \ tail)$$

Si $tail$ est nil, il est omis et l'on obtient une liste de structures musicales associées à leur date d'occurrence *time-stamp*.

8.4.1.2. *Exemples*

1) La pièce structurée p du tableau 8.2 peut se représenter par la structure musicale suivante :

$$(p_1 \ @ \ -20 \ p_2 \ @ \ -5 \ p_3 \ @ \ 50)$$

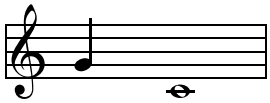
2) La structure musicale représentant le canon à deux voix du tableau 8.3 est la suivante :

$$(p \ @ \ 0 \ p \ @ \ d), \ d > 0$$

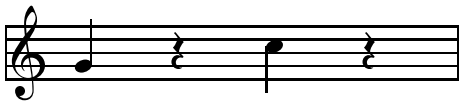
3) Si $[p1, d]$, $[p2, d]$ et $[p3, d]$ sont les notes de l'accord du tableau 8.4, sa structure musicale est la suivante :

$$([p1, d] \text{ @ } 0 \ [p2, d] \text{ @ } 0 \ [p3, d] \text{ @ } 0)$$

4) Soit les pièces suivantes et leur représentation sous forme de structures musicales :

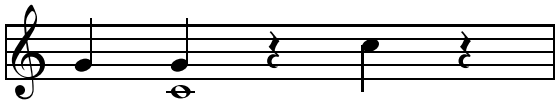


$$p_1 = \left([(g4, p), \tfrac{1}{4}] \text{ @ } 0 \ [(c4, p), 1] \text{ @ } \tfrac{1}{4}\right)$$



$$p_2 = \left([(g4, p), \tfrac{1}{4}] \text{ @ } 0 \ [REST, \tfrac{1}{8}] \text{ @ } \tfrac{1}{4} [(c4, p), \tfrac{1}{4}] \text{ @ } \tfrac{3}{8} \ [REST, \tfrac{1}{8}] \text{ @ } \tfrac{5}{8}\right)$$

Alors la pièce :



peut se représenter par la structure musicale suivante :

$$p_3 = (p_1 \text{ @ } 0 \ p_2 \text{ @ } \tfrac{1}{4})$$

8.4.2. *Opérateurs structuraux*

On peut construire des structures musicales plus complexes au moyen d'opérateurs variés.

La *concaténation horizontale* est notée « $-$ » et consiste à mettre bout à bout les occurrences musicales. Plus précisément :

$$p_1 - p_2$$

est défini par :

$$(p_1 \text{ @ } 0 \ p_2 \text{ @ } d_1)$$

où d_1 est la durée de p_1. La *concaténation verticale* est notée « | » et consiste à faire démarrer les occurrences en même temps. Plus précisément :

$$p_1 \mid p_2$$

est défini par :

$$(p_1 \ @ \ 0 \ p_2 \ @ \ 0)$$

8.4.3. *Représentations graphiques des structures musicales*

Il est classique d'associer à un terme fonctionnel une représentation arborescente dont les nœuds correspondent aux opérateurs du terme et les feuilles correspondent aux constantes et aux variables. Par exemple, la figure 8.8 montre la représentation arborescente de l'expression arithmétique $(a + 1) \times 3$.

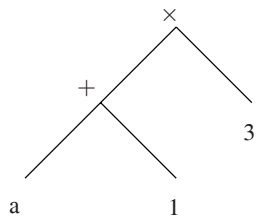


Figure 8.8. *Representation arborescente d'une expression arithmétique*

En fait, la représentation n'est pas forcément purement arborescente. Par exemple, considérons le terme suivant :

$$(S \ @ \ 0 \ S \ @ \ d)$$

Graphiquement, celui-ci peut se représenter par un arbre ou par un DAG, c'est-à-dire un graphe orienté acyclique (voir figure 8.9). Dans le premier cas, le nœud S est dupliqué alors que dans le deuxième, il est partagé. Concernant la représentation interne, le partage d'un nœud n'a pas d'incidence sur le mécanisme de calculs du terme. Son seul intérêt consiste en un coût moindre en termes de place mémoire.

Dans le cas de la musique, une telle représentation n'est pas toujours intuitive quand l'arbre prend de l'ampleur car la dimension temporelle se trouve noyée dans les opérateurs. Pour cette raison, selon les cas à illustrer, nous utiliserons aussi une représentation sous forme de boîtes imbriquées. Une structure musicale est alors représentée par un rectangle. Si l'opérateur principal définissant la structure musicale est une concaténation horizontale (respectivement une concaténation verticale), alors la boîte est partagée par un trait vertical (respectivement un trait horizontal). Quand l'opérateur principal est la concaténation musicale, on peut se référer au formalisme graphique de Mira Balaban [BAL 94] que nous n'avons la place de développer ici.

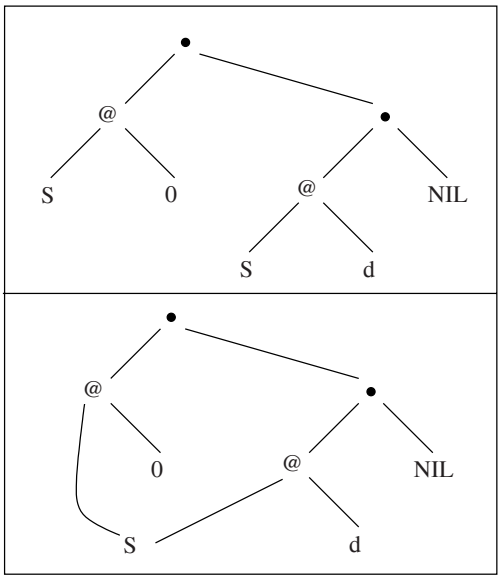


Figure 8.9. *Un même terme représenté sous forme d'arbre et de DAG*

8.4.4. *Extraction d'informations à partir des termes fonctionnels*

L'extraction d'informations se fait au moyen de procédures récursives très simples permettant un calcul de bas en haut. Il suffit, en chaque nœud, d'appliquer une fonction permettant d'évaluer les enfants, puis de réaliser le calcul du père en fonction de ses enfants.

Prenons pour exemple l'opérateur *eval-start* qui calcule la liste des points temporels de début de toutes les occurrences d'une pièce. Ainsi, la pièce P du tableau 8.2 fournit la liste suivante en résultat :

$$\textit{eval-start}(P) = (-30\ 0\ 50)$$

Considérons une représentation ne comportant que les attributs temporels, faisant abstraction des autres informations musicales. Une pièce peut alors se représenter par une liste comportant les éléments suivants :

1) l'attribut *start*,

2) l'attribut *clip*,

3) la liste des occurrences internes de la pièce.

Si la liste des occurrences de la pièce est non vide, alors la pièce est composée ; sinon, elle est élémentaire. Une occurrence peut alors se définir par une liste composée des éléments suivants :

1) la pièce,

2) l'attribut *time-stamp*.

Par exemple, la pièce P est représentée par la liste :

$$(30\ 80\ ((-10\ 55)\ -20)\ ((5\ 55)\ -5)\ ((0\ 30)\ 50))$$

Soit une occurrence p constituée des attributs *start* et *time-stamp* ; la fonction *eval-start (p)* doit donc réaliser les calculs suivants :

1) si p est élémentaire : *start* + *time-stamp* ;

2) si p est composée, soit l la liste des résultats de l'application de *eval-start (p)* à toutes ses occurrences ; il faut réaliser le décalage en ajoutant *time-stamp* à tous les éléments de l.

La fonction *eval-start* suivante, écrite en scheme, prend pour argument une occurrence et réalise le calcul des positions temporelles de ses occurrences élémentaires.

eval-start.scm

```
 1:  ; Une pièce est une liste de la forme :
 2:  ; (start clip occurrence1 occurrence2 ...)
 3:  ;
 4:  ; Une occurrence est une liste de la forme :
 5:  ;     (p time-stamp)
 6:  ; où p est une pièce
 7:  ;
 8:  ; Accesseurs
 9:
10:  (define start car)           ; prend en argument une pièce
11:                               ; renvoie l'attribut start
12:  (define clip cadr)           ; prend en argument une pièce
13:                               ; renvoie l'attribut clip
14:  (define occurrences cddar)   ; prend en argument une occurrence
15:                               ; renvoie la liste de ses occurrences
16:  (define time-stamp cadr)     ; prend en argument une occurrence
17:                               ; renvoie sa date d'occurrence
18:  (define piece car)           ; prend en argument une occurrence
19:                               ; renvoie la pièce
20:
21:  (define (eval-start occurrence)
22:    (cond ((null? (occurrences occurrence))
23:           (list (+ (start (piece occurrence))
24:                    (time-stamp occurrence))))
```

```
25:          (else (map (lambda(x) (+ x (time-stamp occurrence)))
26:                     (map-append eval-start
27:                                 (occurrences occurrence))))))
28:
29:  (define (map-append f l)
30:    (apply append (map f l)))
31:
32:
33:
```

eval-start.scm

Voici un exemple de session avec trace sous l'interprète guile. Dans cet exemple, la pièce p a été un peu développée de façon à mieux illustrer le mécanisme récursif.

```
guile> (define p '((-30 80
                        ((-10 55) -20)
                        ((5 55
                            ((0 10) 5)
                            ((0 45) 10))
                         -5)
                        ((0 30) 50))

                   0))
guile> (trace eval-start)
(eval-start)
guile> (eval-start p)
[eval-start ((-30 80 (# -20) (# -5) (# 50)) 0)]
|  [eval-start ((-10 55) -20)]
|  (-30)
|  [eval-start ((5 55 (# 5) (# 10)) -5)]
|  |  [eval-start ((0 10) 5)]
|  |  (5)
|  |  [eval-start ((0 45) 10)]
|  |  (10)
|  (0 5)
|  [eval-start ((0 30) 50)]
|  (50)
(-30 0 5 50)
```

Cette fonction réalise le calcul d'un seul opérateur. Il est pratique de la généraliser en une fonctionnelle prenant en argument le calcul à réaliser sur les pièces élémentaires et celui à réaliser sur les pièces composées afin de supporter tous les opérateurs. De cette façon, le code est plus court dès qu'il y a plusieurs opérateurs, car il est factorisé.

eval-general.scm

```
 1:  (define (evaluation eval-feuille eval-noeud p)
 2:    (define (eval-interne p)
 3:      (let ((liste-occurrences (occurrences p)))
 4:        (if (null? liste-occurrences)
 5:            (eval-feuille p)
 6:            (eval-noeud p
 7:                        (map-append eval-interne
 8:                                    liste-occurrences)))))
 9:    (eval-interne p))
10:
11:  (define (eval-start-feuille p)
12:    (list (+ (start (piece p)) (time-stamp p))))
13:
14:  (define (eval-start-noeud p l)
15:    (map (lambda(x) (+ x (time-stamp p))) l))
```

eval-general.scm

Dans le cas qui nous intéresse, le calcul est alors mis en œuvre par l'appel suivant :

```
guile> (evaluation eval-start-feuille eval-start-noeud p)
```

8.4.5. *Structures musicales et formes musicales*

Les structures musicales que nous avons introduites jusqu'ici sont des termes clos, c'est-à-dire sans variables. En effet, les arguments terminaux des opérateurs sont des constantes. L'introduction de variables dans cette algèbre permet d'exprimer des abstractions correspondant à des formes musicales. En encadrant le terme ainsi obtenu par une lambda-abstraction, on obtient une forme pouvant s'appliquer à divers arguments effectifs. Par exemple, reprenons le terme correspondant au canon à deux voix :

$$(P \ @ \ 0 \ P \ @ \ d)$$

En remplaçant les symboles P et d par des paramètres, on peut exprimer la forme musicale au moyen de la fonction suivante :

$$Canon2(X, D) = (X \ @ \ 0 \ X \ @ \ D)$$

Ainsi, pour tout motif musical P et toute durée d, on exprime un canon par l'application suivante :

$$Canon2(P, d)$$

Exemples

1) Une mélodie constituée des notes $e_1, ..., e_n$ peut se décrire par l'expression :

$$-(e_1, e_2, ..., e_n) = e_1 - e_2 - ... - e_n$$

et se représenter par la figure 8.10.

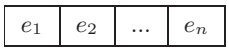


Figure 8.10. *Une mélodie*

2) Un accord constitué des notes $e_1, ..., e_n$ peut se décrire par l'expression :

$$|(e_1, e_2, ..., e_n) = e_1|e_2|...|e_n$$

et se représenter par la figure 8.11.

Figure 8.11. *Un accord*

3) Les pièces p_1 et p_2 du paragraphe 8.4.1.2 peuvent s'écrire :

$$p_1 = ([(g4, p), \tfrac{1}{4}] - [(c4, p), 1])$$
$$p_2 = ([(g4, p), \tfrac{1}{4}] - [REST, \tfrac{1}{8}] - [(c4, p), \tfrac{1}{4}] - [REST, \tfrac{1}{8}])$$

8.4.6. *Discussion*

Dans ce paragraphe, nous allons, au moyen d'exemples, mettre en évidence les qualités, les défauts et les limites de ce formalisme. Dans un premier temps, nous explorons la sémantique des opérateurs utilisés pour structurer les pièces ; dans un deuxième temps, nous illustrons l'aspect hiérarchique.

8.4.6.1. *Sémantique des opérateurs*

8.4.6.1.1. La concaténation musicale

Reprenons notre motif numéro 1 représenté sur la figure 8.1 et considérons le même découpage en notes. Ce motif peut se décrire au moyen de la concaténation musicale par le terme suivant :

$$MOTIF.CM = ([(a\flat 5, p), \tfrac{1}{4}] \mathbin{@} 0\ [(g5, p), \tfrac{1}{4}] \mathbin{@} \tfrac{1}{4}\ [(f5, p), \tfrac{1}{4}] \mathbin{@} \tfrac{1}{2}\ [(g5, p), \tfrac{1}{4}] \mathbin{@} \tfrac{3}{4}\ [(d5, p), \tfrac{1}{4}] \mathbin{@} 1\ [(e\flat 5, p), \tfrac{1}{4}] \mathbin{@} \tfrac{5}{4}\ [(b\natural 4, p), \tfrac{1}{4}] \mathbin{@} \tfrac{3}{2}\ [(c5, p), \tfrac{1}{4}] \mathbin{@} \tfrac{7}{4})$$

Considérons le terme $MOTIF.CM.3$ correspondant au terme $MOTIF1.CM$ privé de la troisième occurrence de note, c'est-à-dire $[(f5, p), \tfrac{1}{4}] \mathbin{@} \tfrac{1}{2}$:

$$MOTIF.CM.3 = ([(a\flat 5, p), \tfrac{1}{4}] \mathbin{@} 0\ [(g5, p), \tfrac{1}{4}] \mathbin{@} \tfrac{1}{4}\ [(g5, p), \tfrac{1}{4}] \mathbin{@} \tfrac{3}{4}\ [(d5, p), \tfrac{1}{4}] \mathbin{@} 1\ [(e\flat 5, p), \tfrac{1}{4}] \mathbin{@} \tfrac{5}{4}\ [(b\natural 4, p), \tfrac{1}{4}] \mathbin{@} \tfrac{3}{2}\ [(c5, p), \tfrac{1}{4}] \mathbin{@} \tfrac{7}{4})$$

Ce terme représente le motif musical de la figure 8.2 et correspond, en quelque sorte, à la suppression de la troisième note du motif numéro 1. En effet, les représentations arborescentes de ces termes diffèrent seulement par la présence ou l'absence du sous-arbre décrivant la troisième note. L'opération permettant de passer du premier motif au second correspond donc à une suppression de sous-arbre.

On constate que les occurrences d'une structure musicale exprimée au moyen de la concaténation musicale sont totalement indépendantes les unes des autres, comme dans le cas des systèmes linéaires de notes de la première section. La différence entre les deux systèmes apparaît aussitôt que plusieurs découpages hiérarchiques sont utilisés, comme nous le verrons dans le dernier exemple.

8.4.6.1.2. La concaténation horizontale

Le motif numéro 1 s'exprime aussi de la façon suivante au moyen de l'opérateur de concaténation horizontale :

$$MOTIF1.CH = ([(a\flat 5, p), \tfrac{1}{4}] - [(g5, p), \tfrac{1}{4}] - [(f5, p), \tfrac{1}{4}] - [(g5, p), \tfrac{1}{4}] - [(d5, p), \tfrac{1}{4}] - [(e\flat 5, p), \tfrac{1}{4}] - [(b\natural 4, p), \tfrac{1}{4}] - [(c5, p), \tfrac{1}{4}])$$

Considérons le terme $MOTIF1.CH.3$ correspondant au même terme privé de sa troisième occurrence :

$$MOTIF1.CH.3 = ([(a\flat 5, p), \tfrac{1}{4}] - [(g5, p), \tfrac{1}{4}] - [(g5, p), \tfrac{1}{4}] - [(d5, p), \tfrac{1}{4}] - [(e\flat 5, p), \tfrac{1}{4}] - [(b\natural 4, p), \tfrac{1}{4}] - [(c5, p), \tfrac{1}{4}])$$

Cette fois-ci, on constate que les occurrences du terme ne sont pas indépendantes, car la structure musicale décrite par le $MOTIF1.CH.3$ (voir figure 8.12) correspond à celle du $MOTIF1.CH$ dont les occurrences postérieures à la troisième ont été décalées logiquement pour la remplacer. Pourtant, l'opération permettant de passer de la représentation arborescente du terme $MOTIF1.CH$ à celle du terme $MOTIF1.CH.3$ est la même que dans le cas précédent. Il s'agit de la suppression d'un sous-arbre. La différence se situe dans la sémantique des opérateurs temporels utilisés pour structurer le motif.

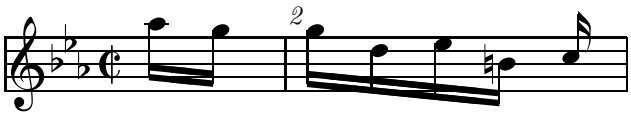


Figure 8.12. *La suppression hiérarchique avec concaténation horizontale implique un décalage logique des occurrences postérieures*

8.4.6.2. *Découpages hiérarchiques*

Le dernier exemple que nous produisons ici est destiné à illustrer l'intérêt des découpages hiérarchiques imbriqués. Contrairement aux systèmes de notes où le découpage est imposé (il s'agit toujours de la note), le système hiérarchique présente plusieurs découpages emboîtés à tous les niveaux de la hiérarchie. De plus, la variété des opérateurs permet aussi de varier la façon de structurer chaque niveau.

Considérons les deux premières mesures de l'*Allemande*, de la deuxième Partita pour piano de J.S. Bach (voir figure 8.4) ainsi que le découpage schématisé sur la figure 8.13, où les symboles représentent les motifs indiqués sur la figure 8.14. Nous allons étudier deux façons de les structurer.

Motif1	Motif2	Motif3
Motif4	Motif5	Motif6

Figure 8.13. *Découpage schématique de l'extrait de l'Allemande de J.S. Bach en six motifs*

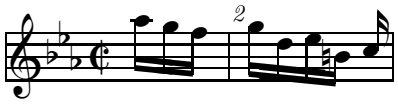
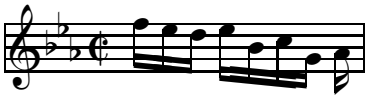
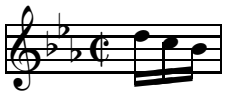
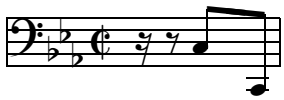
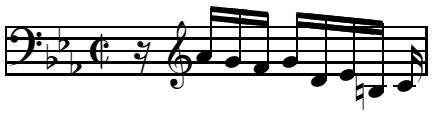
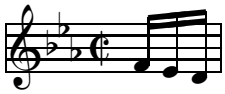
Terme	Motif musical
MOTIF1	
MOTIF2	
MOTIF3	
MOTIF4	
MOTIF5	
MOTIF6	

Figure 8.14. *Définition des six motifs composant l'extrait de l'Allemande de J.S. Bach*

8.4.6.2.1. Structuration horizontale

La forme musicale suivante exprime une structuration temporelle au moyen d'une concaténation verticale de deux concaténations horizontales. Nous qualifierons cette forme d'*horizontale* car elle privilégie les relations temporelles mélodiques, comme la suite va le montrer :

$$Horizontale(M1, M2, M3, M4, M5, M6) = (M1{-}M2{-}M3) \mid (M4{-}M5{-}M6)$$

L'extrait considéré peut s'exprimer au moyen de l'application suivante :

$$A.CH = Horizontale(MOTIF1.CH, MOTIF2.CH, MOTIF3.CH, MOTIF4.CH, MOTIF5.CH, MOTIF6.CH)$$

où les symboles apparaissant comme arguments sont définis eux-mêmes au moyen de la concaténation horizontale de notes. Le premier motif a été donné précédemment sous cette forme. Les autres s'expriment exactement de la même façon.

Considérons le terme $A.CH.3$ correspondant à l'application suivante :

$$A.CH.3 = Horizontale(MOTIF1.CH.3, MOTIF2.CH, MOTIF3.CH, MOTIF4.CH, MOTIF5.CH, MOTIF6.CH)$$

Ce terme est défini avec la même application de la même forme, à l'exception du premier argument qui a été substitué par le terme $MOTIF1.CH.3$ qui représente le motif musical de la figure 8.12.

Dans l'exemple précédent, on a constaté que le résultat de cette opération implique un décalage logique des occurrences postérieures de la durée de l'occurrence supprimée. Par conséquent, la durée de l'occurrence globale change. Il en résulte un décalage logique des occurrences postérieures à $MOTIF1.CH.3$ au niveau supérieur. Les motifs 2 et 3 se trouvent donc désynchronisés par rapport à la partie de la main gauche comme l'indique le schéma de la figure 8.15, où le symbole S figure une partie muette.

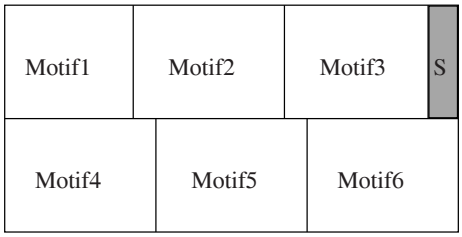


Figure 8.15. *Structuration horizontale : désynchronisation des parties de la main droite*

8.4.6.2.2. Structuration verticale

Afin de préserver les concaténations verticales lors d'une suppression telle que celle que nous considérons, il faut structurer l'extrait différemment. Par exemple, on peut considérer que l'extrait est une concaténation horizontale de trois parties qui sont elles-mêmes des concaténations verticales de deux motifs, comme l'exprime la forme suivante, que nous qualifions par conséquent de verticale :

$$Verticale(M1, M2, M3, M4, M5, M6) = (M1 \mid M4) - (M2 \mid M5) - (M3 \mid M6)$$

L'extrait considéré peut s'exprimer au moyen de l'application suivante :

$$A.CV = Verticale(MOTIF1.CH, MOTIF2.CH, MOTIF3.CH, MOTIF4.CH, MOTIF5.CH, MOTIF6.CH)$$

Considérons, de la même façon que précédemment, le terme $A.CV.3$ correspondant à l'application suivante :

$$A.CV.3 = Verticale(MOTIF1.CH.3, MOTIF2.CH, MOTIF3.CH, MOTIF4.CH, MOTIF5.CH, MOTIF6.CH)$$

Les motifs utilisés en arguments étant toujours définis au moyen de la concaténation horizontale, la durée du premier argument se trouve diminuée, comme dans l'exemple précédent. Mais cette fois-ci, la désynchronisation des motifs suivants ne peut plus se produire au niveau supérieur, car la fin du motif est complétée logiquement par un silence (voir figure 8.16).

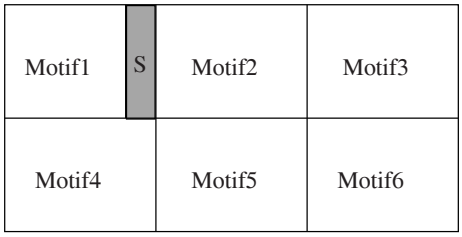


Figure 8.16. *Structuration verticale : synchronisation des parties conservée*

8.4.6.3. *Limites du formalisme*

Nous allons développer les points suivants :

1) la sémantique des opérateurs ne permet pas de définir finement toutes les relations temporelles possibles entre les parties musicales ;

2) l'arborescence de la structure limite le nombre de relations temporelles en chaque nœud ;

3) une seule structure peut s'avérer insuffisante pour le compositeur.

8.4.6.3.1. Limites sémantiques

Dans ce système, le compositeur peut définir des relations temporelles entre les parties en utilisant des opérateurs. Il peut structurer sa pièce en choisissant un découpage à chaque niveau qui correspond aux relations qu'il souhaite conserver durant l'édition.

La concaténation musicale est l'opération la plus générale car elle relie chaque partie avec son parent et ses enfants. Elle n'induit pas de relation temporelle entre les parties sœurs apparaissant à un même niveau. Elle ne supporte pas l'incertitude car elle nécessite la connaissance des dates de début des parties.

La concaténation horizontale associe un nœud parent avec tous ses enfants dans une relation de succession mélodique. Un argument d'une concaténation horizontale se trouve ainsi en relation avec ses parties sœurs. La concaténation horizontale privilégie les relations mélodiques, tandis que la concaténation verticale privilégie les relations harmoniques.

Chaque nœud interne de la hiérarchie pouvant admettre un opérateur temporel quelconque et chaque découpage à chaque niveau étant aussi quelconque, les possibilités de structurer une pièce sont nombreuses. Cependant, les opérateurs utilisés constituent une limite à son pouvoir d'expression. Ainsi, considérons à nouveau la forme $Horizontale$ définie précédemment par la formule :

$$Horizontale(M1, M2, M3, M4, M5, M6) = (M1 - M2 - M3) \mid (M4 - M5 - M6)$$

Aucune relation temporelle ne peut être précisée entre $M2$ et $M5$ au moyen de ce terme. Tous les cas représentés sur la figure 8.17 ainsi que leur symétrique sont possibles. Ces deux motifs peuvent être totalement disjoints, $M2$ se situant avant $M5$ ou le contraire, ils peuvent se chevaucher, commencer ou terminer ensemble. Au moyen de découpages appropriés des motifs eux-mêmes, ces relations peuvent être exprimées pour chaque couple de parties concernées. Chaque motif admet alors un découpage pour chaque relation de ce genre dans laquelle il est impliqué. Avec l'augmentation du nombre de parties et du nombre de relations nécessitant un découpage, le système devient trop lourd.

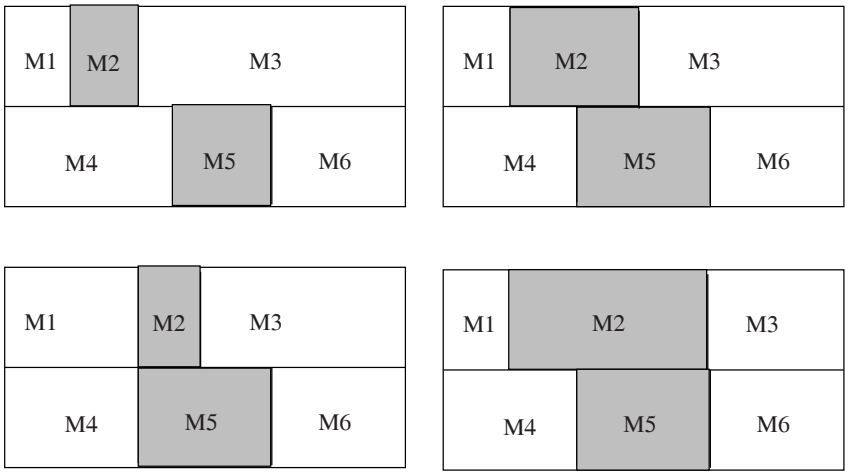


Figure 8.17. *Multiplicité des relations possibles entre M2 et M5*

8.4.6.3.2. Limite structurelle

Il serait possible de pallier l'inconvénient précédent en ajoutant des relations temporelles entre M_2 et M_5. Mais cette pratique conduirait à une représentation hétérogène sous forme de graphe mélangeant les opérateurs et les relations qui sortirait totalement du cadre fonctionnel [DES 02].

8.4.6.3.3. Unicité de la structure

A la lecture des traités musicaux, il s'avère que plusieurs structurations temporelles sont pertinentes pour un même morceau. En effet, comment exprimer au moyen d'une seule structure hiérarchique les relations horizontales liant les éléments mélodiques et les relations verticales liant ces mêmes éléments mélodiques avec les éléments harmoniques simultanés ? Cela est impossible.

De plus, la prise en considération de deux structures simultanées d'une même pièce est assez délicate, compte tenu de la dualité de ces deux genres de relations mises en évidence dans les exemples précédents.

8.4.7. *Les opérateurs rythmiques d'OpenMusic*

OpenMusic est un environnement de programmation visuelle appliqué à la composition musicale. L'organisation temporelle d'une pièce musicale est basée sur celle des structures musicales de Mira Balaban auxquelles a été ajoutée une unité temporelle. Les valeurs temporelles d'OpenMusic sont particulièrement cohérentes avec la notation musicale dans le but de produire des partitions.

OpenMusic est un ensemble de classes écrites en CLOS :

– le *conteneur* correspond à une structure musicale composée ;

– le *conteneur simple* correspond à une structure musicale élémentaire ;

– un *groupe*, une séquence, une superposition, un accord, une polyphonie, une mélodie, une voix, une mesure sont des structures musicales composées particulières ;

– la *note* et le *silence* sont les atomes musicaux de cette représentation.

8.4.7.1. *Combinaison temporelle*

La combinaison temporelle est un peu plus compliquée que chez Mira Balaban car elle fait intervenir la notion de quantum. En effet, le système d'unités de mesure de durées et de points temporels qui est implicite dans les structures musicales fait l'objet d'un attribut supplémentaire local à chaque structure musicale et d'une sémantique appropriée.

Un conteneur comprend les trois attributs temporels suivants :

1) l'attribut q représente une fraction de la noire et définit une unité. Si ce paramètre vaut 2, par exemple, l'unité correspondant est la demi-noire, c'est-à-dire la croche ;

2) l'attribut e sert à exprimer la durée du conteneur dans l'unité définie par q ;

3) l'attribut d correspond à l'attribut *time-stamp* de Mira Balaban. Il sert à exprimer la position temporelle d'un objet. Cette valeur n'a de sens que si l'objet est contenu dans un objet de plus haut niveau. L'unité du déplacement est celle de l'objet le contenant.

On notera dans la suite :

$$p = (q, e, d, p_1, p_2, \ldots, p_n)$$

une structure musicale admettant q, e et d pour attributs temporels et p_1, p_2, ..., p_n pour structures musicales de niveau inférieur. Une telle structure musicale est représentée graphiquement sur la figure 8.18.

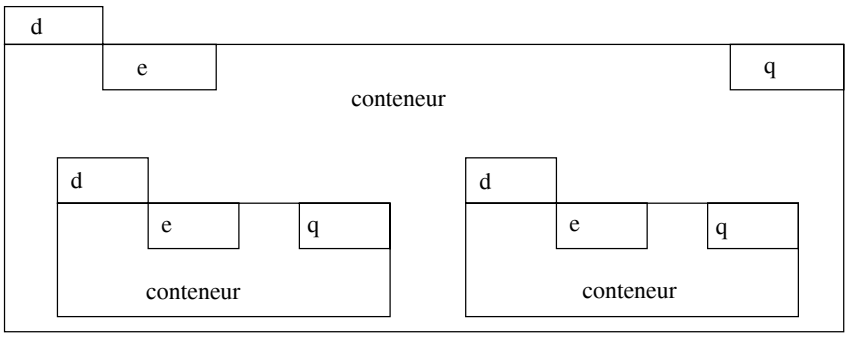


Figure 8.18. *Représentation graphique d'un conteneur*

8.4.7.2. *Opérateurs rythmiques*

Les opérateurs s'appliquent à des conteneurs de façon récursive. Divers opérateurs ont été réalisés dans OpenMusic.

L'*opérateur d'extension et de compression temporelle* – que nous noterons ext – prend en arguments d'une part la structure musicale à transformer et d'autre part un facteur d'étirement exprimé sous la forme d'un rationnel. Soit $p = (q, e, d, p_1, p_2, \ldots, p_n)$ la structure musicale à transformer, n/p le facteur d'étirement et $(q', e', d', p'_1, p'_2, \ldots, p'_n)$ la structure résultante. La sémantique locale de

cet opérateur est la suivante :

$$
\begin{aligned}
&q\text{'} = q \times p \\
&e\text{'} = e \times n \\
&d\text{'} = d \times n \\
&p\text{'}_1 = ext(p_1) \\
&p\text{'}_2 = ext(p_2) \\
&... \\
&p\text{'}_2 = ext(p_2)
\end{aligned}
$$

Cet opérateur s'applique récursivement sur la structure en profondeur d'abord en modifiant les champs q et e des objets et en mettant à jour les différents d des sous-objets des conteneurs (voir figure 8.19).

L'*opérateur de fusion de structures musicales* prend en arguments deux structures musicales et rend comme résultat une structure musicale dans laquelle sont regroupés tous les atomes des structures initiales.

L'*opérateur de masquage* consiste à rendre muette une structure musicale lorsqu'une autre est active. Cet opérateur utilise des calculs réalisés pour la fusion de structures musicales.

8.4.7.3. *Discussion*

L'implémentation des structures hiérarchiques dans OpenMusic est particulièrement bien adaptée au contexte musical par la prise en considération de quantum de temps pouvant varier selon le niveau hiérarchique et rendre compte de l'aspect multiéchelle de la musique. De plus, la nature orientée objets de l'implémentation permet de dériver naturellement par héritage les opérateurs structuraux du plus général (la concaténation musicale) aux plus particuliers, comme la mélodie ou les accords.

8.5. L'approche logique

Dans cette section, nous présentons des relations temporelles pertinentes musicalement permettant d'obtenir un pouvoir d'expression bien plus puissant que celui des opérateurs fonctionnels. Ces relations se situent dans le cadre logique dont nous rappelons les principes au paragraphe 8.5.1. Ensuite, nous présentons les travaux de James F. Allen sur la maintenance de relations temporelles dans le cadre du récit. Enfin, nous évoquons une étude d'Alan Marsden sur les puissances d'expression comparées des deux formalismes.

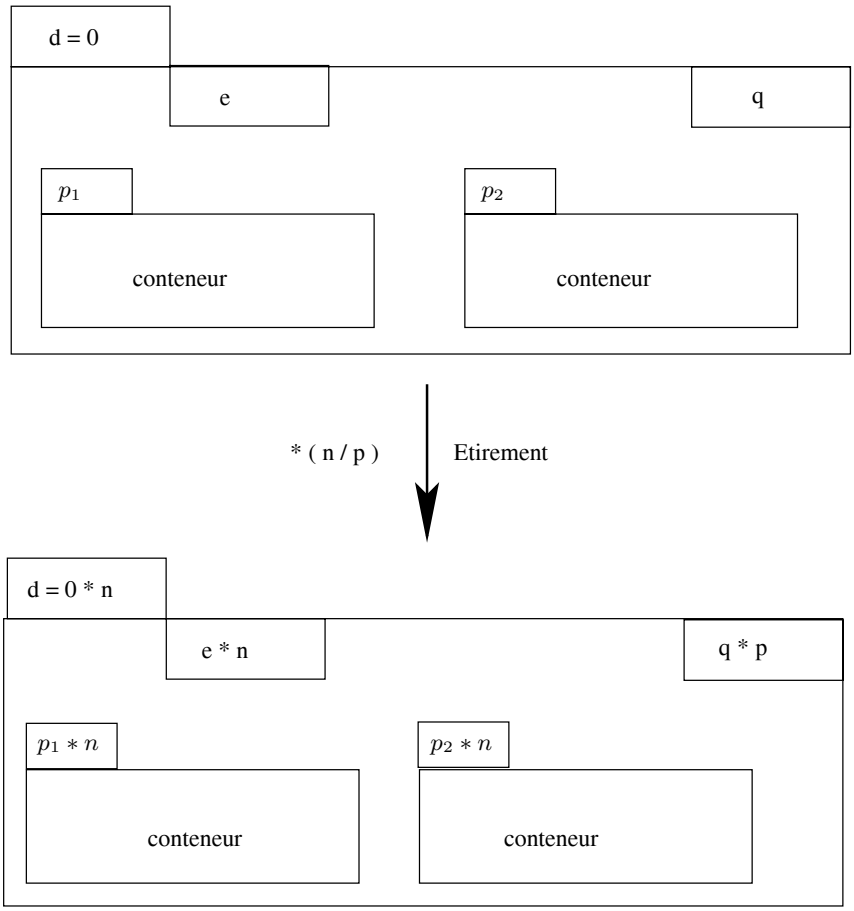


Figure 8.19. *Description locale de l'étirement temporel*

8.5.1. *Le paradigme logique*

La programmation logique se différencie par deux caractères essentiels de la programmation fonctionnelle. En effet, la logique utilise la notion de relation qui étend la notion de fonction de deux façons :

1) le sens du calcul n'est pas orienté *a priori* mais dépend dynamiquement de la configuration des paramètres ;

2) le nombre de résultats d'un calcul, appelés *solutions* dans le cadre logique, n'est pas limité à un seul, mais est quelconque, voire nul et même possiblement infini.

Un ensemble de relations entre des objets peut se représenter sous la forme d'un graphe dont les nœuds sont les objets et les arcs, les relations. Le premier point implique qu'il n'y a pas de direction unique de calcul comme dans le cas fonctionnel, mais que ces directions dépendent des relations utilisées.

Le deuxième point exprime la notion d'indéterminisme. Alors que la fonction produit un résultat déterminé par ses arguments d'appel, la relation doit quant à elle pouvoir admettre plusieurs résultats simultanés.

8.5.2. *La maintenance de relations temporelles*

La représentation temporelle de James F. Allen [ALL 83] est intéressante musicalement pour plusieurs raisons. Premièrement, elle est basée sur la notion d'intervalle temporel, ce qui convient particulièrement aux événements musicaux, car ils ont généralement une durée et sont rarement ponctuels. Deuxièmement, les relations sont explicitées et permettent un raisonnement. Enfin, elle permet d'exprimer l'imprécision, ce qui est particulièrement utile dans le cadre de l'analyse et de l'aide à la composition.

8.5.2.1. *Les relations temporelles entre intervalles*

Treize relations suffisent à décrire toutes les possibilités de positions relatives entre deux intervalles en l'absence d'information sur la durée. Elles sont nommées et dénombrées sur la figure 8.20.

8.5.2.2. *Raisonnement sur les relations temporelles*

Les relations sont maintenues dans un réseau dont les nœuds sont les intervalles. Chaque arc est étiqueté par l'ensemble des relations possibles entre les deux intervalles qu'il relie. En cas d'incertitude, toutes les relations sont associées à l'arc, ce qui ne pose pas de problème car les relations sont disjointes.

Quand une nouvelle relation est entrée dans le réseau, toutes ses conséquences sont calculées par clôture transitive. Par exemple, si le fait $i\ during\ j$ est ajouté alors que $j\ before\ k$ est dans le réseau, alors le fait $i\ before\ k$ est inféré. Ce nouveau fait est ajouté de la même façon au réseau, en introduisant éventuellement lui aussi de nouvelles conséquences. Cet algorithme est en $O(n^2)$ en temps et en espace, où n est le nombre de nœuds.

8.5.2.3. *Notation*

La description d'un réseau d'événements en relation est l'ensemble des descriptions des arcs du réseau. Un arc est décrit de la façon suivante :

$$A\ (r_1 r_2\ ...\ r_n)\ B$$

Relation	**Symbole**	**Symbole de l'inverse**	**Schéma**
X before Y	$<$	$>$	XXX YYY
X equal Y	$=$	$=$	XXX YYY
X meets Y	m	mi	XXXYYY
X overlaps Y	o	oi	XXX YYY
X during Y	d	di	XXX YYYYY
X starts Y	s	si	XXX YYYYY
X finishes Y	f	fi	XXX YYYYY

Figure 8.20. *Relations temporelles entre intervalles*

où A et B représentent les extrémités de l'arc, c'est-à-dire les événements en relation, et les r_i sont l'ensemble des relations possibles entre A et B. Formellement, l'arc ci-dessus correspond à la formule logique suivante :

$$A\ r_1\ B \vee A\ r_2\ B \vee ... \vee A\ r_n\ B$$

8.5.3. *Limitations dans le cadre musical*

Alan Marsden a montré les limites du pouvoir d'expression de la représentation temporelle au moyen de l'approche fonctionnelle, par comparaison aux relations d'Allen. Il a également montré les limitations de l'utilisation des relations d'Allen dans le cadre musical en raison des répétitions [MAR 94]. Nous résumons ses résultats dans ce paragraphe.

8.5.3.1. *Limites des opérateurs temporels*

Soit les séquences $A\ n_1\ n_2\ B$ et $X\ n_3\ Z$ associées à deux parties superposées, où n_1, n_2 et n_3 sont des notes. Dans le formalisme fonctionnel, considérons l'expression suivante :

$$(A - n_1 - n_2 - B)\ |\ (X - n_3 - Z)$$

Elle permet seulement d'exprimer que les deux parties commencent et terminent en même temps. L'expression suivante :

$$\big((A - n_1 - n_2) \mid (X - n_3)\big) - (B \mid Z)$$

exprime de surcroît que les notes B et Z commencent et terminent en même temps. Mais cette expression ne permet pas de restreindre l'indéterminisme sur l'ordre des notes n_1, n_2, n_3.

En utilisant les relations d'Allen, on peut écrire :

$A\ (m)\ n_1$
$A\ (<)\ n_2$
$A\ (<)\ B$
$A\ (s = si)\ X$
$A\ (<\ m\ o)\ n_2$
$A\ (<)\ Z$
$X\ (<\ m\ o\ fi\ di)\ n_1$
$X\ (<\ m\ o)\ n_2$
$X\ (<)\ B$
$X\ (m)\ n_3$
$X\ (<)\ Z$
$n_1\ (m)\ n_2$
$n_1\ (<)\ B$
$n_1\ (<\ m\ o\ s\ d)\ n_3$
$n_1\ (<)\ Z$
$n_3\ (fi = f)\ n_2$
$n_3\ (m)\ B$
$n_3\ (m)\ Z$
$n_2\ (m)\ B$
$n_2\ (m)\ Z$
$B\ (=)\ Z$

L'indéterminisme sur les notes n_1, n_2, n_3 peut être restreint. Par exemple, pour exprimer que la note n_3 ne peut pas commencer avant la note n_1, on pourrait écrire les relations suivantes :

– $A\ (s =)\ X$: A commence en même temps que X et peut se terminer avant ou après X ;

– $A\ (< m)\ n_2$: A se termine avant que n_2 commence ;

– X $(< m\ o\ fi\ di)$ n_1 : X se termine avant que n_1 ne termine ou en même temps ;

– X $(< m\ o)$ n_2 : X se termine avant que n_2 ne termine ;

– n_1 $(< m\ o\ s)$ n_3 : n_1 se termine avant que n_3 ne termine et peut commencer avant ou en même temps ;

– n_3 $(fi = f)$ n_2 : n_3 et n_2 finissent en même temps.

Ces relations sont inexprimables au moyen d'une expression fonctionnelle. En effet, la structure arborescente de l'expression fonctionnelle limite le nombre de relations associées aux nœuds. Un événement est associé en tant qu'argument à un seul opérateur et, de même, il peut être associé en tant que résultat à un seul opérateur.

8.5.3.2. *Représentation de la répétition*

Les formes musicales avec répétition ne sont pas exprimables au moyen des relations d'Allen. Par exemple, dans la forme musicale ABA, la première occurrence de A est en relation m avec B, alors que la deuxième admet la relation mi avec B.

8.6. Les systèmes hiérarchiques basés sur les grammaires

La hiérarchie temporelle est aussi reliée à la notion de grammaire. En effet, une expression temporelle est un mot appartenant à un langage algébrique décrit par une grammaire. Une hiérarchie peut alors être vue comme un arbre de dérivation de cette grammaire. Dans cette correspondance, l'application d'une fonction f à des arguments a_1, a_2, ..., a_n est associée à une réécriture d'un symbole non terminal par une règle de production. Ainsi, la *sémantique* des fonctions peut être formellement définie par le biais des grammaires attribuées [BAR 93]. Le pouvoir d'expression se trouve augmenté d'une part, par la possibilité d'ajouter des calculs supplémentaires en chaque nœud concernant des dimensions musicales autres que celle du temps, et d'autre part, les calculs ne sont pas uniquement orientés de bas en haut, mais aussi de haut en bas.

Dans cette section, nous rappelons la définition des grammaires attribuées et nous présentons leur intérêt et leurs limitations dans le cadre musical.

8.6.1. *Sémantique temporelle de hiérarchies musicales*

Les grammaires attribuées sont des grammaires algébriques de mots dans lesquelles des attributs sont attachés à chaque production de la grammaire. Ces attributs sont reliés entre eux au sein de chaque production par un calcul défini par une *sémantique*. Ce calcul peut être orienté de bas en haut (membre droit vers membre gauche) ou de haut en bas (membre gauche vers membre droit). Les attributs sont typés selon leur sens de calcul. Dans le premier cas, on parle d'*attributs synthétisés* et dans le

deuxième, d'*attributs hérités*. Durant le processus d'analyse, l'arbre de dérivation est construit et les attributs sont calculés incrémentalement selon la sémantique des productions.

Dans le cadre de l'aide à la composition musicale, l'arbre de dérivation de la grammaire est considéré comme une structure de données qui se construit petit à petit durant le processus de composition. A chaque étape, le compositeur donne une valeur à un attribut ou instancie une nouvelle production, ce qui induit des calculs incrémentaux des attributs de l'arbre. En effet, à la différence des systèmes fonctionnels dont la syntaxe des termes permet d'indiquer des valeurs uniquement sur les feuilles, les arbres de dérivation associés aux grammaires attribuées permettent d'indiquer des valeurs n'importe où dans l'arbre, ce qui conduit à un mécanisme d'évaluation plus compliqué.

Supposons que le compositeur indique une valeur d'instrument sur une feuille correspondant à une note de sa hiérarchie. Cette valeur doit remonter dans l'arbre pour signaler cet instrument aux niveaux hiérarchiques supérieurs, c'est-à-dire aux parties qui contiennent cette note. Par contre, si le compositeur indique une valeur d'instrument sur un nœud interne de l'arbre correspondant à une partie musicale, cela signifie pour lui que tous les éléments de cette partie sont écrits pour cet instrument.

Il en découle un mécanisme d'évaluation bidirectionnel. Pour tout nœud de l'arbre, chaque valeur peut soit remonter dans l'arbre, soit descendre vers les feuilles. Cette évaluation s'effectue au moyen d'une double récursivité quand la grammaire n'est pas circulaire.

L'interface permettant au compositeur de modifier l'état de la hiérarchie n'est pas abordée ici. Nous considèrerons simplement que lors de l'ajout d'une production, les attributs qui lui sont attachés sont initialisés à une valeur notée $\perp$ signifiant qu'ils ne sont pas définis. Cette valeur est utilisée dans l'expression de la sémantique des productions.

8.6.1.1. *Définition des grammaires attribuées*

Une grammaire attribuée est un quadruplet $G = < N, T, P, ATT >$ où :

1) N est un alphabet non terminal ;

2) T est un alphabet terminal ;

3) ATT est l'ensemble des symboles d'attributs typés ;

4) P est un ensemble de règles de production qui sont des triplets $< mg, md, s >$, où $mg \in F$, $md \in (A \bigcup F)^*$ et s est la sémantique associée à la production, soit un ensemble d'équations fonctionnelles sur les attributs de la production. Les attributs d'une production sont notés $a(E_i)$, où $E \in A \bigcup F$, $a \in ATT$ et E_i est la i-ième occurrence de la lettre E dans la production (le rang *zéro* étant réservé au

membre gauche). Alors, une équation fonctionnelle est une égalité dont le membre gauche est un attribut de la production et le membre droit une expression dont les arguments sont des attributs de la production.

8.6.1.2. *Exemple*

Soit G_0 une grammaire pouvant être utilisée pour calculer divers attributs musicaux, tels que le volume, la mesure ou l'instrument, d'une séquence musicale :

– $N = \{S\}$;

– $T = \{-\}$, où « $-$ » représente l'opérateur de concaténation horizontale (voir paragraphe 8.4.1) ;

– $ATT = \{V_1, V_2\}$ avec V_1 et V_2 appartenant à l'ensemble des valeurs de volume augmenté de la valeur indéfinie $\perp$, par exemple $\{\perp, pp, p, mp, mf, f, ff\}$. De plus, V_1 est un attribut hérité alors que V_2 est synthétisé. L'utilité de cette distinction sera expliquée ultérieurement ;

– P contient les productions suivantes :

$$
\begin{array}{ll}
S \longrightarrow S - S & V_2(S_0) = V_2(S_1)\ U\ V_2(S_2) \\
 & \text{si } V_1(S_0) \neq \perp \\
 & \text{alors } V_1(S_1) = V_1(S_0),\ V_1(S_2) = V_1(S_0) \\
S \longrightarrow (p, v, d) & V_2(S_0) = \{v\} \\
S \longrightarrow (REST, d) & V_2(S_0) = \emptyset
\end{array}
$$

Le symbole S_0 représente le non-terminal S qui est membre gauche de la règle et les symboles S_1 et S_2 représentent les premières et deuxièmes occurrences du non-terminal S dans le membre droit de la règle. On associe à chaque production de la grammaire un graphe de dépendance de ses attributs. La figure 8.21 représente le graphe de la production de P.

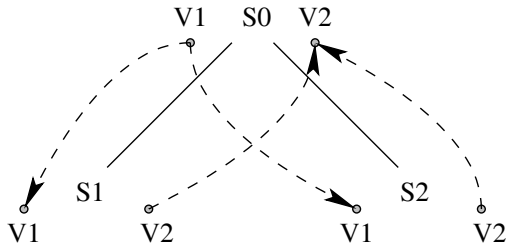


Figure 8.21. *Graphe de dépendance de la production de G_0*

8.6.1.2.1. Remarque

La grammaire proposée dans cet exemple ne comporte pas de symboles terminaux autres que l'opérateur. En effet, dans le cadre de l'aide à la composition, une hiérarchie peut être augmentée à la fois par les feuilles (par la dérivation de la feuille, c'est-à-dire son remplacement par une structure temporelle) et par la racine (par l'utilisation de la hiérarchie dans un autre contexte). Même une feuille représentant une simple note définie par sa hauteur, durée, volume et timbre peut à tout moment être remplacée, par exemple, par une broderie autour de cette note. Pour cette raison, la notion de pièce vide de Mira Balaban, qui correspond à la notion de symbole terminal dans ce formalisme, disparaît. Dans un autre contexte nécessitant de nommer des parties de pièces par des symboles dans un ensemble B, il faut ajouter à l'ensemble des productions P les règles suivantes :

$$\forall e \in T \setminus Op,\ q_e : E \longrightarrow e$$

où Op représente l'ensemble des opérateurs, $\{-\}$ dans notre exemple.

8.6.2. *Evaluation*

On distingue deux formes différentes d'évaluation. L'évaluation statique consiste à évaluer complètement tout un arbre de dérivation à la fois. L'évaluation incrémentale [HUD 91] s'effectue à partir d'un arbre et d'une modification d'un attribut de l'arbre. Elle consiste à ne calculer que les informations de l'arbre qui sont amenées à changer suite à cette modification. Logiquement, une évaluation incrémentale donne le même résultat qu'une évaluation statique de l'arbre modifié. Mais elle est plus efficace en temps et elle permet d'implémenter un mécanisme de traitement des conflits [BAR 94]. Dans ce paragraphe, nous présentons essentiellement l'évaluation statique.

Si la grammaire est non circulaire, c'est-à-dire si les dépendances des attributs se représentent par un graphe sans cycle, l'évaluation d'une hiérarchie est particulièrement simple. Dans [BAR 93], on trouve une grammaire destinée au calcul de l'attribut représentant le mode d'une pièce. Cette grammaire étant circulaire, elle est un peu compliquée et ne constitue pas un bon exemple préliminaire. Le lecteur pourra s'y référer ultérieurement.

Prenons pour exemple la grammaire G_0. Les deux attributs V_1 et V_2 représentent deux façons de calculer une même donnée V. Pour fixer les idées, considérons qu'il s'agit des instruments pour lesquels la pièce est écrite. Ces deux attributs permettent de modéliser les deux mécanismes de calcul de la donnée V qui peut être héritée, c'est-à-dire dépendant de son contexte extérieur, ou synthétisée, c'est-à-dire dépendant de son propre contenu. Ces deux attributs sont donc calculés de façon indépendante. Dans le cas normal, un seul des deux est utilisé à la fois. Si les deux attributs d'un même

nœud prennent une valeur, un conflit survient si les deux valeurs sont différentes. Par souci de clarté, nous nous restreignons au cas normal.

L'évaluation se fait au moyen d'une double récursivité permettant de calculer les attributs synthétisés en remontant et les attributs hérités en descendant. Pour la mettre en œuvre, il est commode de procéder au préalable à un étiquetage des nœuds de l'arbre afin de discerner ceux dont l'attribut est synthétisé de ceux dont il est hérité. Pour cela, on peut utiliser deux étiquettes de non-terminaux, E pour les nœuds qui ont une valeur ou qui sont synthétisés et S pour les nœuds qui sont hérités. L'ensemble de productions s'élargit à tous les cas possibles (voir figure 8.22).

P1	$E \longrightarrow E - E$	$V_2(E_0) = V_2(E_1)\ U\ V_2(E_2)$
P2	$E \longrightarrow E - S$	$V_2(E_0) = V_2(E_1)$ $V_1(S_1) = V_1(E_0)$
P3	$E \longrightarrow S - E$	$V_2(E_0) = V_2(E_1)$ $V_1(S_1) = V_2(E_0)$
P4	$E \longrightarrow S - S$	$V_1(E_0)$ is set $V_1(S_1) = V_1(E_0)$ $V_1(S_2) = V_1(E_0)$
P5	$S \longrightarrow S - S$	$V_1(S_1) = V_1(S_0)$ $V_1(S_2) = V_1(S_0)$

Figure 8.22. *Productions discernant l'attribut synthétisé de l'attribut hérité*

Le programme scheme suivant utilise deux procédures mutuellement récursives : une procédure $Eval$ d'évaluation locale en un nœud et une procédure $EvalNode$ propageant l'évaluation vers les enfants. La procédure $Eval$ correspond à la sémantique de la production dont le nœud est racine. La procédure $EvalNode$ consiste à lancer $EvalNode$ sur tous les attributs synthétisés des enfants, puis $Eval$ sur le nœud courant et enfin $EvalNode$ sur tous les attributs hérités des enfants.

Une hiérarchie est représentée par sa racine. Chaque nœud est représenté par une liste composée de quatre informations : l'étiquette (E ou S), l'attribut V_1, l'attribut V_2 et la liste des enfants. Il est à noter que dans la sémantique, les attributs sont des ensembles. Ici, nous les représentons par des listes en gardant la multiplicité des valeurs.

eval-attribute.scm

```
1:  ; noeud n : (label :  E (set or synthesized) ou S (inherited)
2:  ;            i1  inherited attribute
2:  ;            i2  synthesized attribute
4:  ;            children)
```

```
 5:
 6:  ; Fonctions d'accès en lecture et en écriture
 7:
 8:  (define label car)
 9:  (define i1 cadr)
10:  (define i2 caddr)
11:  (define children cadddr)
12:
13:  (define (set-i1! n value)
14:    (let ((address (cdr n)))
15:      (set-car! address value)))
16:
17:  (define (set-i2! n value)
18:    (let ((address (cddr n)))
19:      (set-car! address value)))
20:
21:  ; Mise en forme des listes d'enfants
22:
23:  (define (synthesized children)
24:    (filter (lambda (n) (eq? (label n) 'E))
25:            children))
26:
27:  (define (inherited children)
28:    (filter (lambda (n) (eq? (label n) 'S))
29:            children))
30:
31:  ; Prédicats
32:
33:  (define bottom? null?)
34:  (define (feuille? n) (null? (children n)))
35:
36:  ; Evaluation
37:
38:  (define (eval! n)
39:    (map eval! (synthesized (children n)))
40:    (eval-node! n)
41:    (map eval! (inherited (children n))))
42:
43:  (define (eval-node! n)
44:    (let ((i1-value (i1 n))
45:          (i2-value (i2 n)))
46:      (if (not (feuille? n))
47:          (set-i2! n (map-append i2 (synthesized (children n)))))
48:      (if (not (bottom? i1-value))
49:          (map (lambda (n)
50:                 a(set-i1! n i1-value))
51:               (inherited (children n))))))
```

```
52:
53:
54:  ; Fonctions auxiliaires générales
55:  (define (map-append f l)
56:    (apply append (map f l)))
57:
58:  (define (filter test? liste)
59:    (cond ((null? liste) '())
60:          ((test? (car liste)) (cons (car liste)
61:                                     (filter test? (cdr liste))))
62:          (else  (filter test? (cdr liste)))))
```

eval-attribute.scm

Voici un exemple de session avec l'interprète guile.

```
guile> (define arbre4 '(E () ()
                          ((E () ()
                              ((E () (forte) ())
                              (E () (piano) ())))
                           (S () () ()))))
guile> (eval! arbre4)
(E () (forte piano)
   ((E () (forte piano)
       ((E () (forte) ())
        (E () (piano) ())))
    (S () () ())))
```

8.6.3. *Grammaires attribuées relationnelles*

Une grammaire attibuée relationnelle se différencie d'une grammaire attribuée ordinaire par la nature de la sémantique de ses productions [COU 88, DER 84]. Ces sémantiques sont des relations exprimées dans une certaine logique, plutôt que des équations fonctionnelles. Aussi, les types des attributs (synthétisés ou hérités) dus à l'orientation des équations fonctionnelles n'ont plus lieu d'être, chaque attribut pouvant être calculé grâce à ses enfants ou à ses frères ou parents.

Grammaire attribuée de concaténations horizontale et verticale

Nous présentons ici la grammaire attribuée relationnelle correspondant aux opérations de concaténations horizontale et verticale présentées au paragraphe 8.4.1. La sémantique des productions est exprimée par des systèmes d'équations. Soit $G = < N, T, P, ATT >$, où les ensembles sont définis par :

– $N = \{E\}$ est l'ensemble des symboles non terminaux ;

– $T = \{-, \mid\}$ est l'ensemble des opérateurs temporels ;

– $ATT = \{d\}$ est l'ensemble des attributs ;

– P est l'ensemble des productions représentées sur la figure 8.23. Les symboles « $-$ » et « $|$ » représentent respectivement la concaténation horizontale et verticale. Le symbole E_0 représente le non-terminal E qui est membre droit de la règle, et les symboles E_1 et E_2 représentent les premières et deuxièmes occurrences du non-terminal E dans le membre droit de la règle.

P1	$E \longrightarrow E - E$	$d(E_0) = d(E_1) + d(E_2)$
P2	$E \longrightarrow E \mid E$	$d(E_0) = d(E_1)$ $d(E_1) = d(E_2)$

Figure 8.23. *Les règles de production de la grammaire G*

8.6.4. *Limitations du formalisme*

Les limites de ce formalisme se situent dans l'expression des structures musicales et des relations temporelles.

8.6.4.1. *Structures musicales*

Les grammaires algébriques attribuées relationnelles ne permettent de représenter que des hiérarchies purement arborescentes. Ceci constitue une limitation non négligeable en pratique car la répétition, en musique, est assez courante.

L'évaluation bidirectionnelle ne fonctionne que dans le cas purement arborescent. Si l'évaluation d'un arbre de dérivation s'exprime fort simplement au moyen de procédures récursives locales à chaque nœud, l'évaluation sur un DAG ne peut s'envisager que plus globalement. En effet, la solution en un nœud situé sur un cycle dépend potentiellement de tous les nœuds du cycle. Une détection du cycle et une résolution globale du système associé à l'ensemble du cycle est nécessaire par des techniques de résolution de contraintes.

8.6.4.2. *Relations temporelles*

Les relations temporelles exprimables au moyen des grammaires attribuées relationnelles définies avec les opérateurs de concaténation horizontale et verticale sont de deux catégories [BAR 98] :

1) les *relations temporelles locales* correspondent aux opérateurs temporels et relient un nœud avec ses enfants ;

2) les *relations temporelles induites* par la structure, déduites par transitivité à partir des relations de la première catégorie.

Parmi les relations locales, on trouve les relations d'Allen m, b, s, d, f, e (voir paragraphe 8.5.2) ainsi que leurs inverses. L'ensemble des relations induites comprend vingt relations d'Allen dont la relation o est absente, de même qu'un certain nombre de disjonctions comprenant la relation o.

Le pouvoir d'expression peut être augmenté au moyen d'attributs de date et de durées, ce qui permet de mélanger les relations qualitatives exprimées par les opérateurs et les relations quantitatives exprimées par les valeurs de dates et de durées.

8.7. Conclusion

Nous avons parcouru plusieurs systèmes de représentation temporelle utilisés en musique. Les systèmes fonctionnels sont certainement les plus simples à mettre en œuvre et fournissent une puissance d'expression satisfaisante dans bien des cas. Les systèmes basés sur les grammaires d'attributs fournissent des calculs incrémentaux confortables, mais cet avantage se paie par une puissance de représentation moindre. Les systèmes logiques sont très puissants en ce qui concerne la définition des relations temporelles, mais n'offrent pas, comme les autres systèmes, un moyen pratique de structurer les données.

Le problème de l'interface n'a pas été abordé. Il est pourtant d'une importance capitale. En effet, que faire d'un système si l'on ne peut le manipuler de façon dynamique dans le cadre de la composition musicale ?

L'interface d'un système linéaire est simple à imaginer car il n'offre pas beaucoup de possibilités : c'est la représentation en pistes que l'on trouve dans les séquenceurs.

L'interface d'un système fonctionnel est composée de fonctions et d'applications de fonctions. Une interface visuelle peut lui être associée, comme dans le logiciel *patchwork* développé à l'IRCAM. Mais son inconvénient principal est qu'elle ne permet que des éditions au niveau des termes et non au niveau de leur résultat. En effet, quand le résultat de l'évaluation d'un terme ne convient pas, il faut modifier le terme de façon à ce que son résultat devienne convenable. Il serait parfois plus pratique pour le compositeur de modifier le résultat directement. Cette correspondance entre le terme et son résultat est difficile à établir. Il en est de même pour les systèmes à base de grammaires attribuées. Les éditions hiérarchiques qu'ils fournissent peuvent s'avérer contraignantes et le compositeur peut parfois souhaiter voir sa musique « à plat » comme une partition. Dans ce contexte, les interactions entre les éditions « à plat » et les éditions hiérarchiques ne sont pas encore totalement résolues.

Malgré les désavantages que nous avons pointés dans la section 8.1, les systèmes à base de pistes sont très répandus et très utilisés. Ce phénomène s'explique en partie par les limitations et la rigidité des systèmes hiérarchiques fonctionnels (ou basés

grammaire). Dans les premiers, seule l'édition des atomes (notes ou sons) est possible, alors que dans les seconds, seules les éditions dans le cadre des structures sont possibles. Les éditions bas-niveau sur les atomes, même si elles sont pénibles dans certains cas (en particulier lors de synchronisations), offrent en contrepartie une liberté totale. Un système qui mixerait de façon adéquate les formalismes existants afin de pouvoir bénéficier de leurs avantages est à bâtir. Il faut pouvoir disposer des avantages des structures hiérarchiques avec leurs dates relatives, des contraintes temporelles linéaires à la Allen pour préciser les relations entre les composants musicaux, et de la liberté d'édition au niveau atomique. Mais ce domaine est encore à l'état de recherche.

8.8. Bibliographie

[ALL 83] ALLEN J.F., « Maintaining knowledge about temporal intervals », *Communications of the ACM*, vol. 26, n° 11, p. 832-843, 1983.

[BAL 89a] BALABAN M., « The cross fertilization relationship between music and ai », *Interface*, vol. 18, p. 89-115, 1989.

[BAL 89b] BALABAN M., « Music structures: A temporal-hierarchical representation for music », *Musicometrika*, vol. 2, p. 1-50, 1989.

[BAL 91] BALABAN M., Music structures: The temporal and hierarchical aspects in music, Rapport technique FC-035 MCS-327, Departement of Mathematics and Computer Science, Université Ben-Gurion, Beer-sheva, 84-105, Israël, 1991.

[BAL 93] BALABAN M., SAMOUN C., « Hierarchy, time and inheritance in music modeling », *Languages of Design*, vol. 1, n° 3, p. 147-172, 1993.

[BAL 94] BALABAN M., ELHADAD M., On the need for visual formalisms for music processing, Rapport technique FC-94-14, Université Ben-Gurion, 1994.

[BAL 98] BALABAN M., MURRAY N., « Interleaving time and structure », *Computers and Artificial Intelligence*, vol. 17, n° 1, p. 1-34, 1998.

[BAR 93] BARBAR K., DESAINTE-CATHERINE M., MINIUSSI A., « The semantics of musical hierarchies », *Computer Music Journal*, vol. 17, n° 4, p. 30-37, décembre 1993.

[BAR 94] BARBAR K., BEURIVÉ A., DESAINTE-CATHERINE M., « On the resolution of musical conflicts », dans *Proceedings of the Conference on New Music Research, Gent, Belgique*, octobre 1994.

[BAR 98] BARBAR K., BEURIVÉ A., DESAINTE-CATHERINE M., « Structures hiérarchiques pour la composition assistée par ordinateur », dans Chemillier M., Pachet F. (dir.), *Recherches et applications en informatique musicale*, Hermès, Paris, 1998.

[COR 94] CORMEN T., LEISERSON C., RIVEST R., *Introduction à l'algorithmique*, Dunod, Paris, 1994.

[COU 88] COURCELLE B., DERANSART P., « Proofs of partial correctness for attribute grammars with application to recursive procedures and logic programming », *Information and Computation*, vol. 2, n° 2881, p. 127-145, 1988.

[DER 84] DERANSART P., Validation des grammaires d'attributs, Thèse de doctorat, Université de Bordeaux I, 1984.

[DES 02] DESAINTE-CATHERINE M., BEURIVÉ A., « Time modeling for musical composition », dans *Proceedings of the FSKD'02, Singapour*, novembre 2002.

[HUD 91] HUDSON S.E., « Incremental attribute evaluation: A flexible algorithm for lazy update », *ACM Transactions on Programming Languages and Systems*, vol. 13, n° 3, p. 315-341, juillet 1991.

[MAR 94] MARSDEN A., The representation of temporal relations in music, non publié, 1994.

Chapitre 9

Algorithmes et techniques pour l'analyse musicale

9.1. Introduction

L'interprétation d'un phénomène musical complexe suppose d'être capable, consciemment ou non, de le décomposer et d'établir des liens entre ses éléments. La notion de s*tructure musicale* est prise ici dans le sens de l'organisation attribuée à *une surface musicale*[1] en termes de parties et de relations/fonctions entre elles à divers niveaux de description. L'analyse musicale vise la révélation de ces descriptions structurelles, souvent d'un point de vue perceptif/cognitif (Plié, 1980).

On emploie souvent des méthodologies informatiques pour étudier les processus analytiques musicaux *en eux-mêmes* et également pour construire des outils d'analyse utilisables dans des applications musicales pratiques comme l'exécution interactive, la composition assistée par ordinateur, la recherche et l'indexation de données musicales, l'expressivité de l'exécution automatique, etc. Dans ce dernier cas, le principal argument est que plus on vise des applications informatiques musicales sophistiquées, plus le système informatique doit « comprendre » les structures musicales.

Chapitre rédigé par Emilios CAMBOUROPOULOS et Pierre-Yves ROLLAND.

1. La surface musicale est considérée comme « le plus bas niveau de représentation ayant une signification musicale » [JAC 87, p. 219]. Par commodité, elle est considérée ici comme une séquence de notes discrètes représentées de façon symbolique, sans éléments structurels de plus haut niveau (marques d'articulation, signature rythmique, etc.)

Il existe différents points de vue sur la façon de créer des modèles informatiques pour les tâches musicales. Une distinction entre les stratégies de modélisation est basée sur la quantité de *connaissances* explicites directement intégrée dans la représentation informatique [CON 95] : dans l'approche relevant de l'*ingénierie des connaissances*, la représentation est entièrement codée « manuellement » par le programmeur-théoricien sur la base de connaissances musicales intuitives ou explicites (par exemple dans les manuels d'enseignement de la musique), alors que dans l'approche relevant de *l'induction empirique,* la représentation est élaborée à partir de généralisations sur un ensemble de phénomènes musicaux d'après divers principes fondamentaux généraux et diverses stratégies. Du fait de la diversité de styles et de « langages » de la musique, les chercheurs ont souvent eu recours à la première approche, avec laquelle ils ont construit des systèmes experts complexes pour effectuer des tâches analytiques particulières dans des domaines musicaux spécialisés (par exemple le contrepoint du XVI^{e} siècle dans le style de Palestrina, l'harmonie tonale du XVIII^{e} siècle, l'analyse schenkérienne, l'analyse timbrale du XX^{e} siècle pour la musique atonale, l'harmonie du jazz, des systèmes correspondant à divers genres musicaux traditionnels, etc.). D'autre part, certains chercheurs ont essayé de construire des systèmes inductifs et moins dépendants de styles musicaux spécifiques ; ces approches utilisent souvent des techniques d'apprentissage automatiques, des méthodologies relevant de la théorie de l'information, des principes et modèles cognitifs, etc. Ce chapitre donne une description brève des modèles relevant de la deuxième stratégie, car ils sont habituellement plus généraux et peuvent être appliqués à des styles et idiomes musicaux variés.

Divers modèles informatiques pour des tâches analytiques *mélodiques* seront décrits par une série d'exemples musicaux. La plupart de ces techniques peuvent être adaptées et appliquées aux fichiers musicaux polyphoniques en tenant compte de paramètres complémentaires (par exemple l'harmonie : on considère un passage musical comme une série d'accords) ou si l'on dispose d'algorithmes de séparation de voix permettant de décomposer un passage polyphonique en plusieurs « mélodies » indépendantes (voir l'aperçu donné dans [TEM 01]).

En supposant que la mélodique est présentée sous la forme d'une séquence de notes ayant des valeurs nominales, les modèles informatiques présentés ici répondent aux questions suivantes :

a) comment détecter les frontières locales pour commencer à segmenter une mélodie ?

b) comment détecter les structures accentuelles et métriques ?

c) comment détecter des patterns musicaux significatifs ?

d) quel est l'effet de la similitude musicale sur la segmentation mélodique ?

e) comment organiser les segments musicaux en catégories « significatives » (par exemple motifs et thèmes) ?

Ce chapitre a pour but de présenter des stratégies permettant de résoudre ces problèmes d'analyse musicale, non de signaler les meilleurs algorithmes (il existe peu d'algorithmes suffisamment testés et peu d'études comparatives, car la musicologie informatique en est toujours à un stade peu avancé). Pour illustrer le propos de façon simple, il illustre l'application d'une série d'algorithmes d'analyse à une mélodie simple, mentionne d'autres méthodes possibles et fournit quelques exemples supplémentaires.

9.2. Frontières locales

Les principes de la Théorie de la forme (*Gestalttheorie*) relativement à l'organisation de la perception, sont un ensemble de règles pratiques portant sur les modes préférentiels de groupement des événements (notamment visuels) en schémas de plus grande échelle. Selon deux des principes de *Gestalt* relativement aux données de bas niveau, les objets proches (principe de Proximité) ou similaires (principe de similitude) tendent à être perçus comme des groupes. Ces principes sont à la base de divers modèles récents de détection des frontières locales dans les séquences mélodiques.

Par exemple, dans le modèle de Tenney et Polansky [TEN 80], les principes de Proximité et de Similitude sont pris comme des descriptions différentes du même phénomène, à savoir un maximum local de la distance entre événements musicaux consécutifs pour n'importe quel paramètre musical, par exemple la hauteur et l'instant initial de la note. Une mélodie donnée est représentée par une séquence d'intervalles de hauteur (en demi-tons) et par une séquence d'intervalles entre instants initiaux (exprimés en multiples de la plus petite unité de durée adéquate ; dans l'exemple ci-dessous il s'agit de doubles-croches). On calcule alors la somme de chaque paire correspondante de données dans les deux séquences de valeurs, et les maximums locaux de la série de sommes sont considérés comme les frontières les plus probables (figure 9.1b).

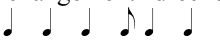
Le modèle de règles de regroupement locales de Lerdahl et Jackendoff [LER 83] essaie lui aussi de repérer les maximums locaux dans les séquences d'intervalle paramétriques. Ce modèle propose des règles plus variées, mais elles sont binaires (aucune valeur ne leur est associée), par exemple :

– *GRP2b* : un intervalle entre instants initiaux est perçu comme une frontière locale s'il est plus grand que le précédent et que le suivant ;

– *GPR3a* : un intervalle de hauteur est perçu comme une frontière locale s'il est plus grand que le précédent et que le suivant ;

– *GPR3d* : dans une séquence de quatre notes, on perçoit une frontière entre les deux notes centrales si les deux premières et les deux dernières notes ont la même durée si les deux notes du milieu ont des durées différentes.

Une règle complémentaire *d'intensification* indique que si plusieurs règles de détail locales s'appliquent dans une position donnée, la frontière locale correspondante est plus marquée. Un exemple est donné dans la figure 9.1a (où seules interviennent les trois règles ci-dessus). Ce modèle a été très influent ; beaucoup d'applications informatiques ont adopté et adapté les règles de Lerdahl et Jackendoff pour la segmentation (par exemple [TEM 01]). Ce modèle, évalué expérimentalement, s'est avéré très fiable [DEL 87].

Selon Cambouropoulos [CAM 96, CAM 97], bien que cette formalisation des principes de la Gestalttheorie (un grand intervalle situé entre de petits intervalles) fournisse le principal facteur de découverte des frontières locales, une approche plus générale se devrait de représenter *n'importe quel* changement d'échelle des intervalles. Par exemple, dans la séquence de durées  , un auditeur entend facilement un point possible de segmentation alors qu'aucun des deux modèles précédents ne suggère de frontière.

Le modèle de segmentation que nous proposons et que nous appelons modèle de détection de frontières locales (MDFL, en anglais *Local Boundary Detection Model LBDM*) est basé sur deux règles : la règle de changement d'identité et la règle de proximité. La règle de changement d'identité est plus simple que tous les principes de la Gestalttheorie, car elle est applicable à partir de deux entités (c'est-à-dire qu'elle permet de juger si deux entités sont identiques ou non), tandis que la règle de proximité exige au moins trois entités (deux entités sont jugées plus proches ou plus similaires que deux autres entités) :

– *règle de changement d'identité (RCI)* : les frontières peuvent être présentées sur n'importe lequel de deux intervalles consécutifs si ces intervalles sont différents. Si les deux intervalles sont identiques, aucune frontière n'est suggérée ;

– *règle de proximité (RP)* : si deux intervalles consécutifs sont différents, la frontière présentée sur le plus grand intervalle est proportionnellement plus forte.

Le modèle de détection de frontières locales suggère *toutes* les positions possibles des frontières locales sur une surface musicale, avec divers degrés de relief. Les valeurs sont normalisées sur l'intervalle [0,1]. Les sommets locaux sont les points les plus susceptibles d'être perçus comme des frontières locales (voir l'exemple de la figure 9.1a et la description formelle de l'algorithme en annexe, A.1). Une description détaillée de l'algorithme est donnée dans [CAM 01a]. Le *MDFL*, appliqué à de nombreuses mélodies, a fourni de très bons résultats. Une évaluation de ce modèle est présentée dans [BAT 98 ; HOT 02].

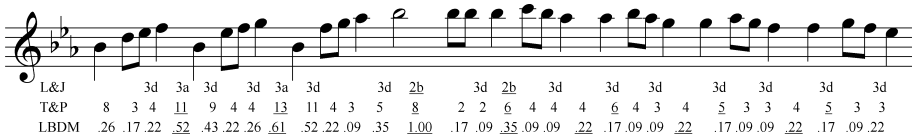


Figure 9.1. *Trois patterns de frontières locales appliqués à la mélodie initiale de la* Valse op. 18 *de Chopin : a) règles de regroupement local de Lerdahl et Jackendoff (en omettant les liaisons et les règles de dynamique) ; b) score des frontières locales de Tenney et Polanski ; c) score des frontières locales selon le MDFL. Les valeurs soulignées indiquent les frontières locales probables.*

Les frontières locales ne suffisent pas à segmenter une surface musicale, comme le montrera la section 9.5. Cependant, elles ont un rôle important dans la segmentation et elles peuvent aussi être utilisées efficacement pour la détermination de l'accentuation et de la structure métrique ; ceci est discuté dans la section suivante.

9.3. Accents locaux et structure métrique

Les accents locaux reflètent la prégnance avec laquelle les notes sont perçues au niveau local : les notes sont généralement plus « accentuées » si elles ont des durées plus longues, si elles sont dans un registre extrême, si elles sont soulignées dynamiquement ou harmoniquement plus importantes, etc. Le mètre est considéré comme une structure abstraite constituée d'un maillage régulier de points à divers niveaux hiérarchiques (voir figure 9.2). La structure métrique est mise en correspondance avec la structure d'accentuation d'une surface musicale et on choisit la « meilleure » combinaison. Divers modèles peuvent servir à déterminer les niveaux d'accentuation et la structure métrique [LEE 91 ; LON 82 ; POV 85 ; ROS 92 ; STE 77 ; TEM 01] mais ils ne seront pas illustrés ici car c'est un sujet qui exigerait un chapitre spécifique.

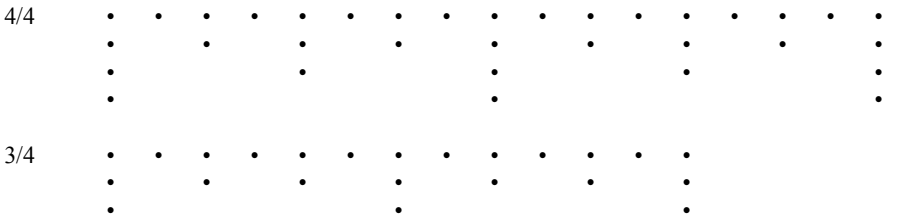


Figure 9.2. *Exemples de structures métriques hiérarchiques*

Nous nous bornerons à exposer une nouvelle hypothèse : que le regroupement local et les structures d'accentuation locales n'ont pas à être des composants indépendants d'une théorie du rythme musical, mais qu'ils sont étroitement associés, de telle façon que si l'un défini, l'autre peut en être déduit automatiquement [CAM 97]. Cette hypothèse se base sur l'observation que les frontières de regroupement sont étroitement liées aux événements accentués entre lesquelles elles se situent. Une frontière perçue dans un continuum donné indique que les éléments qui le délimitent sont plus saillants que d'autres événements situés plus loin. Epstein écrit : « En fait, la démarcation revient à l'accent : c'est l'accent exigé à ce moment pour tracer la frontière d'un segment temporel » [EPS 95, p. 24].

Par exemple, en attribuant des scores de frontière locale (par exemple, des valeurs *MDFL*) à toutes les paires d'intervalles successifs, on obtient la structure d'accentuation locale de la surface. Les valeurs de cette séquence de niveaux d'accentuation indiquent la prégnance relative des notes au niveau local.

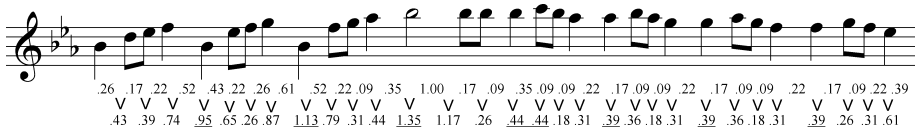


Figure 9.3. *Niveaux d'accentuation locaux pour la mélodie initiale de la* Valse *de Chopin* op. 18

La forte corrélation étroite entre le groupement et les structures d'accentuation est importante car elle permet d'élaborer un modèle qui n'a pas besoin de deux méthodes distinctes pour la détection des frontières locales et pour les phénomènes d'accentuation. Contrairement au modèle de Lerdahl et Jackendoff (figure 9.4a), notre modèle relie directement la structure d'accentuation locale à la structure de regroupement (figure 9.4b), ce qui permet de créer plus facilement et plus efficacement des systèmes informatiques de traitement du rythme musical.

a. Structure de regroupement/structure métrique/ structure d'accentuation.

b. Structure de regroupement/structure d'accentuation ↔ structure métrique.

Figure 9.4. *a) La théorie du rythme de Lerdahl et Jackendoff ; b) notre théorie*

Une fois que la structure d'accentuation locale a été définie, on peut essayer de lui faire correspondre une structure métrique bien formée ; on trouvera des détails sur les modèles d'appariement de patterns dans [PAR 94]. Cet appariement peut porter sur plusieurs niveaux métriques hiérarchisés de battements rythmiques, ou sur un niveau unique, voire, probablement, le « niveau zéro », selon le type de musique.

Le score total d'accentuation, correspondant à un réseau métrique donné, peut être calculé en ajoutant simplement les accents de tous les événements dont le début coïncide avec les points du réseau. Si entre les divers(es) positions/déplacements d'un réseau métrique on trouve une valeur totale « significativement » plus grande, on considère que c'est le meilleur choix. Si les divers placements d'un réseau reçoivent des valeurs similaires, on peut penser qu'il y a ambiguïté métrique dans le réseau considéré. Par exemple, on peut tenter de faire correspondre des réseaux métriques en 2/4 et en 3/4 avec les accents de l'exemple mélodique.

Dans cet exemple, on suppose que le score d'accentuation de la première note est de 1. La somme de tous les accents de chaque réseau est telle que les sommes sont comparables. Le tableau 9.1 donne la valeur totale pour chaque déplacement de chaque réseau, multipliée par la valeur de réseau métrique lui-même * (pour que les sommes soient comparables). La meilleure correspondance est obtenue pour le réseau métrique 3/4 qui commence sur la première note de la mélodie. La valeur totale est calculée comme suit :

$$(1{,}00 + 0{,}95 + 1{,}13 + 1{,}35 + 0{,}44 + 0{,}39 + 0{,}39 + 0{,}39) \times 3/4 = 4{,}53$$

Ce réseau métrique est justement le rythme indiqué par le compositeur dans la partition.

1**La structure métrique peut être déduite de la structure d'accentuation, mais, en même temps, elle influence fortement et résout l'ambiguïté de la structure de groupement/accentuation. Des accents métriques peuvent être ajoutés aux niveaux d'accentuation de façon à obtenir une régulation de la structure de groupement d'un morceau.

L'hypothèse de bas niveau présentée ici ne couvre que les traits structuraux de bas niveau de l'organisation en termes d'accentuation et de groupement. Il est possible qu'aux niveaux plus élevés d'organisation ces structures soient partiellement indépendantes, voire en conflit. Il est en tous cas très intéressant de

1. Le mètre n'est pas simplement un phénomène mental suscité par la musique. Il a un statut psychologique autonome dans un contexte culturel donné et il influence la façon dont la musique est exécutée et perçue. Clarke décrit une expérience qui illustre l'influence de différentes structures métriques sur l'exécution d'une même mélodie [CLA 85].

voir tout ce que représente et permet d'inférer une structure de groupement locale bien définie (par opposition aux structures d'accentuation et métriques locales considérées indépendamment).

Réseau métrique	2/4		3/4		
Décalage	0	1/4	0	1/4	2/4
Valeur totale	3,01	3,95	4,53	2,39	3,11

Tableau 9.1. *Découverte de la structure métrique préférée (valeur soulignée) pour le profil d'accentuation de la*

9.4. Patterns musicaux

Un des principaux facteurs de compréhensibilité de la musique est l'autoréférence, c'est-à-dire le réseau de relations de nouveaux passages musicaux avec ce que l'on a déjà entendu. La répétition structurelle et la similitude sont des dispositifs cruciaux pour l'établissement de ces relations. Les entités musicales similaires sont organisées en catégories musicales comme motifs rythmiques et mélodiques, thèmes et variations, groupes de progression harmonique, etc. (section 9.6). La similitude musicale établit des rapports entre entités musicales différentes, mais permet en premier lieu de définir ces entités en contribuant directement à la segmentation d'une surface musicale en unités significatives (section 9.5).

On emploie souvent les techniques de traitement de patterns pour déterminer la similitude musicale. On fait, en général, l'hypothèse qu'il est possible de considérer une surface musicale comme une chaîne d'entités musicales (notes, accords, etc.) auxquelles on peut appliquer des techniques de reconnaissance de formes ou d'induction. Dans ce chapitre, le terme d'*induction de patterns* désigne les techniques qui permettent l'extraction des patterns utiles d'une chaîne, et celui de *reconnaissance de formes* renvoie aux techniques servant à trouver tous les cas d'un pattern prédéterminé dans une chaîne donnée. Les deux algorithmes d'induction de patterns indiqués aux paragraphes 9.4.1 et 9.4.2 peuvent être utilisés pour l'analyse mélodique automatisée. On trouvera des aperçus de l'application des algorithmes de traitement de patterns aux chaînes musicales dans [CAM 99, CRA 98, ROL 99]. ****note**

2. Il existe de nombreux algorithmes d'appariement de chaînes, qui sont généralement appliqués aux chaînes de caractères ou aux chaînes biologiques (par exemple, l'ADN ou les chaînes de protéines). Quelques-uns sont discutés dans [APO 85] et dans [CRO 94].

Généralement, les entités constituant un pattern musical n'ont pas la même prégnance : certaines notes (ou accords, etc.) ressortent davantage que d'autres en termes de position métrique, de durée, de registre, d'harmonie, de hiérarchies tonales, etc. Quel type de techniques d'appariement de formes est le plus adapté pour établir des ressemblances entre des chaînes structurées, comme des passages mélodiques ? Plus précisément, bien que l'appariement approximatif semble être la solution évidente pour saisir les variations musicales (par exemple le remplissage et « l'amincissement » du matériel thématique, des changements rythmiques, des changements de hauteur, des changements tonaux, etc.), l'appariement exact permet-il de traiter ce phénomène ?

Pour simplifier, considérons qu'il existe deux grandes approches :

a) les techniques d'appariement approximatif de patterns (AAP) appliquées à la surface musicale non structurée ;

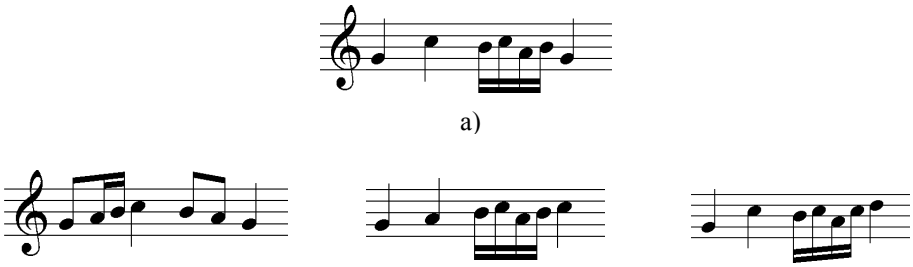
b) les techniques d'appariement exact de patterns (AEP) appliquées sur la surface musicale et à diverses réductions de celle-ci en composants structurellement plus saillants.

La première approche est basée sur l'idée que les segments musicaux considérés comme parallèles (similaires) partagent *certains* éléments identiques (par exemple, deux occurrences d'un motif mélodique auront un nombre « significatif » de notes ou d'intervalles communs, mais pas nécessairement tous). Plusieurs algorithmes d'AAP sont décrits par exemple dans [BLO 85, COP 90, ROL 96, ROW 95, STA 93]. La deuxième approche est basée sur l'idée que des segments musicaux parallèles soient nécessairement identiques du point de vue d'au moins un profil paramétrique de la surface ou d'une réduction de celle-ci (par exemple, deux occurrences d'un motif mélodique auront le même profil paramétrique au niveau superficiel ou à un certain niveau plus abstrait comme des patterns de notes/intervalles métriquement forts ou tonalement importants). On trouvera des techniques informatisées basées sur cette approche dans [CAM 98a, HIR 97].

Quels sont les avantages et les inconvénients de chacune de ces méthodologies ? Pour prendre un exemple, considérons les segments mélodiques tonaux de la figure 9.5. Quel est le degré de similitude des segments *b, c et d* avec le segment *a* ? Supposons que chaque segment mélodique est représenté sous la forme d'une séquence de notes décrites par deux valeurs : la hauteur et l'instant initial (bas de la figure 9.5).

Pour l'AAP, chacun des segments *b, c, d* est identique à 71 % au segment *a* car cinq de leurs sept notes correspondent. Selon le seuil choisi, les trois segments mélodiques ont la même similitude (et la même dissimilitude) avec *a*. Pourtant, il est

clair pour un musicien que le segment *b* est (dans la plupart des contextes tonaux) bien plus similaire à *a* que les autres car *a* et *b* se correspondent de la « bonne » façon : les notes principales sont identiques, et on ignore les « ornementations », moins importantes.



segment a : [g, 0], [c, 4], [b, 8], [c, 9], [a, 10], [b, 11], [g, 12]
segment b : [g, 0], [a, 2], [b, 3], [c, 4], [b, 8], [a, 10], [g, 12]
segment c : [g, 0], [a, 4], [b, 8], [c, 9], [a, 10], [b, 11], [c, 12]
segment d : [g, 0], [c, 4], [b, 8], [c, 9], [a, 10], [c, 11], [d, 12]

Figure 9.5. *Degré de similitude des segments mélodiques* b, c *et* d *pour la segmentation de* a

La deuxième méthode d'appariement de formes exige un prétraitement significatif ; par exemple, au lieu de se contenter d'examiner les segments mélodiques au niveau superficiel, on construit des niveaux abstraits de représentation qui reflètent les propriétés structurelles des segments mélodiques (par exemple des notes plus longues, des notes métriquement plus fortes ou des notes tonalement importantes). Il faut cependant souligner qu'il est possible de tenir compte dans les techniques d'AAP de la prégnance structurelle en attribuant des *poids* aux appariements des éléments des patterns, par exemple leur contribution à la similitude pour chaque transformation, notamment relativement à la durée et aux intervalles de hauteur, comme l'ont proposé et mis en œuvre Mongeau et Sankoff [MON 90] et Rolland [ROL 99].

La seconde méthode a sur la première, l'avantage de donner des raisons explicites au jugement de similitude : les propriétés communes que l'on découvre sont codées explicitement, et cette connaissance peut être réutilisée dans d'autres tâches d'analyse ou de composition. Par contre, elle exige un traitement supplémentaire pour produire des réductions musicalement significatives de la surface.

9.4.1. *Un algorithme d'induction de patterns basée sur des répétitions exactes*

Un algorithme efficace permettant de calculer toutes les répétitions à l'intérieur d'une chaîne est décrit dans [CRO 81 ; Iliopoulos *et al.*, 1996]. Pour une chaîne donnée de symboles (par exemple la chaîne d'intervalles de hauteur), le processus d'appariement commence par la plus petite longueur de pattern (deux éléments) et se termine lorsqu'il a trouvé l'appariement le plus long. La complexité de cet algorithme est *O (N*log N)*, *N* étant la longueur de la chaîne.

Cet algorithme peut être appliqué à un nombre quelconque de profils paramétriques de la surface mélodique ou de ses réductions (par exemple : intervalles de hauteurs, contours, durées, intervalles entre instants initiaux, écarts de dynamique, etc.).

On voit que cette procédure de découverte de tous les patterns mélodiques, identiques sur un grand nombre de chaînes paramétriques mélodiques, produit un nombre très élevé de patterns possibles, dont la plupart sont peu naturels et non pertinents pour un auditeur, musicien ou analyste humain.

Une procédure a été mise au point pour attribuer une valeur de prégnance à chacun des patterns découverts, d'après les critères suivants :

a) préférer les patterns les plus longs,

b) préférer les patterns les plus fréquents,

c) éviter les chevauchements.

On peut créer une *fonction de choix* qui calcule un score numérique d'un pattern selon ces principes, par exemple :

$$f\ (\mathrm{L}, \mathrm{F}, \mathrm{DOL}) = \mathrm{F}^{a} \cdot \mathrm{L}^{b} / 10^{c} * \mathrm{DOL}$$

où *L* est la longueur de modèle ; *F* la fréquence des occurrences dans un pattern ; *DOL* le degré de chevauchement ; et *a, b, c* des paramètres auxquels on peut donner expérimentalement diverses valeurs constantes.

Chaque pattern découvert par cet algorithme reçoit un score suivant la fonction de choix.

Les patterns ayant les meilleurs scores sont considérés comme les plus significatifs (voir figure 9.6).

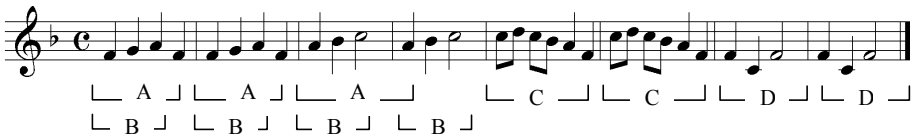


Figure 9.6. Frère Jacques *: principaux patterns de hauteur signalés par l'algorithme d'induction exact et la* fonction de choix *(appliquée aux intervalles diatoniques seulement)*

9.4.2. *Un algorithme d'induction de patterns basée sur des répétitions approximatives*

On l'a vu, l'induction de patterns basée sur des répétitions exactes (IPBRE) peut donner d'excellents résultats. Cependant, en plus des traitements supplémentaires requis pour la réduction de la surface musicale, il peut aussi y avoir des limitations lorsque l'on a affaire à certains genres musicaux. Le jazz improvisé en fait partie. Par exemple, dans le be-bop, genre dominant né dans les années 1940, les techniques massivement utilisées par les improvisateurs telles que l'ornementation, la variation locale et les syncopes, rendent difficile une approche par réduction-induction.

Dans le cadre de l'induction de patterns basée sur des répétitions approximatives (IPBRA), ces phénomènes locaux sont pris en charge en autorisant en assouplissant la contrainte d'égalité entre les différentes occurrences d'un pattern. Un *pattern* est alors défini comme un ensemble de passages (ou *segments*) mélodiques qui sont significativement similaires. Chaque segment est appelé une *occurrence* du pattern. Dans certains cas, on exige que toutes les paires de segments dans un pattern soient significativement similaires, mais nous nous concentrerons ici sur le cas où :

– l'une des occurrences (le *prototype*) est considérée comme la plus représentative du pattern ;

– chaque occurrence est significativement similaire au prototype.

Un tel *pattern en étoile* est illustré sur la figure 9.7 : le pattern apparaît non dans une seule mélodie comme dans l'exemple donné plus haut pour l'IPBRE, mais dans plusieurs (quatre mélodies).

Un certain nombre d'algorithmes d'IPBRA ont été proposés, par exemple dans Cope (1991), Rolland [ROL 99a, ROL 01b], Rowe (1993), Smaill *et al*. (1993), Stammen & Pennycook [STA 93] et Stech (1981). Comme l'expliquent Rolland & Ganascia [ROL 99b], certains d'entre eux peuvent fonctionner en temps réel, les autres étant limités au temps différé.

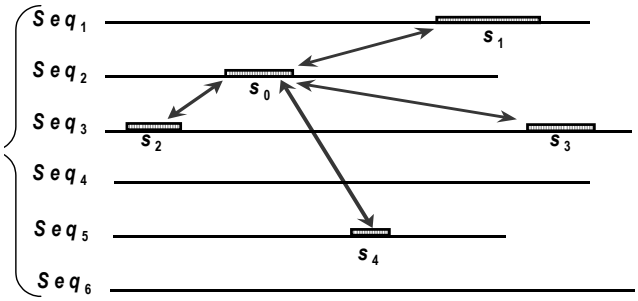


Figure 9.7. *Exemple de pattern en étoile apparaissant dans quatre mélodies. Sa liste d'occurrences est* $\{s_0, s_2 \ldots s_4\}$ *et le prototype est* s_0

Pour formaliser la notion de « significativement similaire », on introduit une *fonction de similitude* (ou, de façon équivalente, une fonction de dissimilitude). Plus spécifiquement, une fonction de similitude normalisée *Simil* prend ses valeurs dans l'intervalle réel [0,1], s_1 et s_2 étant 2 passages mélodiques, $Simil(s_1, s_2)$ est d'autant plus proche de 1 (resp. de 0) que leur similitude calculée est forte (resp. faible). Par définition, un couple de passages (s_1, s_2) est significativement similaire, ou *équipollent*, si et seulement si $Simil(s_1, s_2)$ est supérieur ou égal à un seuil prédéfini, appelé seuil d'équipollence.

Toute fonction de similitude est basée sur un *modèle de similitude* qui définit la procédure de comparaison de deux passages. C'est une caractéristique importante de tout algorithme d'IPBRA.

Dans le modèle de similitude le plus simple, on compare successivement chaque note[3] du premier passage à la note correspondante du second passage et la similarité entre les deux passages est mesurée par la proportion de notes concordantes. Cette structure de correspondance fixe (« distance de Hamming ») est très limitative car elle exige que les deux passages aient le même nombre de notes. Elle est utilisée par exemple par le module d'induction de patterns d'EMI (Cope 1991).

A l'autre extrémité du spectre, figurent les modèles de similarité appartenant à la famille de la « distance d'édition ». Ils utilisent des structures de correspondance variables, afin de faire correspondre les notes des deux passages comparés de manière optimale. Une illustration en est donnée en figure 9.9 et figure 9.10, sur la

3. Les silences peuvent être traités de la même façon que les notes. Par souci de simplicité, nous ne parlons ici que des notes.

base du couple de passages de la figure 9.8. Lors de la comparaison des passages s_1 et s_2, une note n de s_1 peut être appariée (mise en correspondance) à une note n' de s_2. Cet appariement est appelé *remplacement-identité* si n est identique à n', et *remplacement-substitution* si elles sont différentes. Mais n peut aussi être appariée :

– à un groupe vide (zéro note) de s_2. Ce cas, dénommé *suppression*, s'interprète comme le fait que s_1 et s_2 seraient plus similaires en enlevant n_1 de s_1 ;

– à un groupe de plusieurs notes $\{n', n''\ldots\}$ de s_2, n'ayant d'ailleurs pas nécessairement la même hauteur ou la même durée que n. Ce cas, la *fragmentation*, s'interprète comme le fait que s_1 et s_2 seraient plus similaires si, dans s_1, n_1 était remplacée par le groupe $\{n', n''...\}$.

De façon analogue on définit d'autres types d'appariements :

– *insertion* (zéro note de s_1 appariée avec 1 note de s_2),

– *consolidation* (l'inverse de la fragmentation),

– *suppression multiple*,

– *insertion multiple*,

– etc.

Chaque appariement est considéré comme contribuant positivement ou négativement à la similitude globale entre s_1 et s_2. Formellement parlant, un nombre réel appelé *contribution* est attribuée à chaque appariement. A l'exception de travaux anciens, la contribution attribuée à un appariement est fonction des caractéristiques de la ou des notes sur lesquelles il porte. Par exemple, la contribution attribuée à la substitution d'une note par une note très similaire (par exemple en termes de hauteur, durée et positions métriques) est plus positive qu'une substitution par une note très différente.

Pour chaque structure de correspondance entre s_1 et s_2 (appelée *alignement*) la somme des contributions de tous les appariements donne un score de similitude. *Simil*(s_1, s_2) est alors défini comme le plus élevé de ces scores, en d'autres termes, comme le score du meilleur alignement possible entre les deux passages.

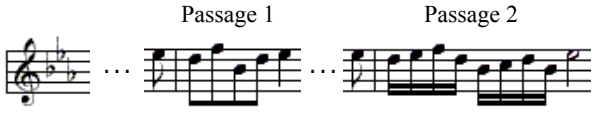


Figure 9.8. *Deux passages mélodiques (J. Haydn,* Concerto pour Trompette en mi bémol majeur*)*

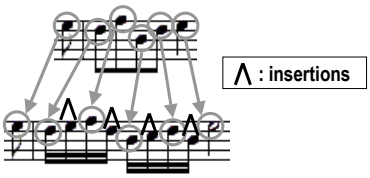


Figure 9.9. *L'un des alignements possibles entre les deux passages, avec six remplacements et quatre insertions*

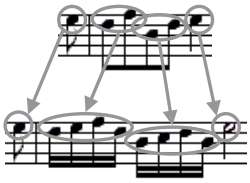


Figure 9.10. *Un autre alignement entre les deux passages, avec deux remplacements individuels et deux remplacements généralisés*

Le calcul de *Simil*(s_1, s_2), prenant en compte tous les alignements possibles, est complexe et fait intervenir une combinatoire énorme. Des techniques de programmation dynamique permettent d'effectuer ce calcul en un temps proportionnel au produit des longueurs de s_1 et s_2.

FlExPat (*Flexible Extraction of Patterns*) est un algorithme d'IPBRA autorisant l'emploi de telles structures de correspondances variables. Une brève description de l'algorithme est donnée en annexe A.3. Pour plus de détails, on pourra se référer aux articles suivants : [ROL 99] pour une perspective musicale, [ROL 01] pour une perspective davantage formelle et technique.

FlExPat comprend deux phases principales :

1) une phase de *construction du graphe d'équipollence* identifie toutes les paires de passages équipollents, c'est-à-dire (rappel) significativement similaires. Les calculs fortement combinatoires de cette phase sont effectués de façon économique (en termes de temps de calcul et de mémoire) grâce à l'emploi de concepts dérivés de la programmation dynamique. Une structure intermédiaire, appelée *graphe d'équipollence,* est produite dans laquelle :

- chaque sommet correspond à (pointe vers) un seul passage et un seul ;
- chaque arête traduit le fait que deux passages sont équipollents. En d'autres termes, en supposant que *v* correspond au passage *s* et que *v'* correspond au passage *s'*, l'arête (*v,v'*) existe si et seulement si *s* et *s'* sont équipollents ;

2) une phase d'*extraction de sous-graphes* extrait les patterns proprement dits à partir du graphe d'équipollence. La forme précise des sous-graphes extraits peut être variable, par exemple :

- des sous-graphes en étoile tels que celui de la figure 9.11 ;
- des cliques maximales comme dans l'algorithme UNSCRAMBLE (voir annexe A.2).

Nous nous focaliserons ici sur le cas de sous-graphes en étoile. Dans chaque sous-graphe extrait, l'ensemble des arêtes représente un ensemble de passages qui est à son tour interprété comme l'ensemble des occurrences d'un pattern en étoile comme défini plus haut. Un prototype est ainsi obtenu pour chaque pattern extrait.

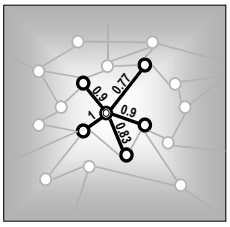


Figure 9.11. *Exemple de sous-graphe en étoile dans le graphe d'équipollence*

Comme pour l'algorithme d'IPBRE, un score est attribué à chaque pattern induit. Il est généralement basé sur les critères suivants :

– favoriser les patterns les plus longs,
– favoriser les patterns ayant le plus d'occurrences.

Il est à noter que, contrairement au cas « exact » où intervenait un troisième critère « éviter les recoupements », FlExPat évite les recoupements de manière plus directe : en effet, lors de la construction du graphe d'équipollence, si deux passages se recoupent trop (le seuil de recoupement étant ajustable par l'utilisateur), leur similarité ne sera pas calculée du tout. On ne risque donc pas de les considérer comme deux occurrences d'un même pattern.

Parmi les principaux paramètres numériques de FlExPat, figure l'entier m_{max} qui contrôle la longueur maximum des passages constitutifs d'un pattern. En pratique, il n'y a pas grand intérêt à rechercher des patterns extrêmement longs ; $m_{max} << N$ où N est le nombre total de notes de la mélodie (ou de l'ensemble de mélodies) où l'on cherche à induire les patterns.

La sortie de FlExPat est une liste de patterns en étoile classés par degrés de force décroissants. Sa complexité temporelle globale théorique est majorée par $O(m_{max}^2.N^2)$, formule que l'on peut réécrire $O(N^2)$ compte tenu de la remarque précédente. Elle est d'ailleurs bien plus faible en pratique.

Nous allons maintenant présenter un exemple de sortie de FlExPat. L'algorithme est exécuté sur un ensemble de dix transcriptions de solos de Charlie Parker sur le schéma d'accord de Blues en do majeur : trois prises (*takes*) de *Cool Blues*, trois prises de *Relaxin' at Camarillo* et quatre prises de *Perhaps*. La longueur totale N est d'environ 2 000. Parmi les valeurs de paramètres de FlExPat, on a fixé m_{max} à 27 et le seuil d'équipollence à 0,7 (avec une fonction de similitude normalisée)

Le graphe d'équipollence a environ 12 000 sommets et 40 000 arêtes. Le plus grand pattern en étoile a trente-quatre segments plus le prototype. Le pattern le plus long a un prototype de longueur vingt-sept (la longueur d'un pattern pouvant être définie comme la longueur de son prototype, ou comme la longueur moyenne de ses occurrences).

La figure 9.12 montre le prototype de l'un des patterns les plus longs (longueur 26) extrait par FlExPat, que nous désignerons dans la suite par Pat1. La figure 9.13, la figure 9.14 et la figure 9.15 montrent ses trois occurrences, classées par similitude décroissante par rapport au prototype, soit 0,93, 0,76 et 0,75.

Pat1 a un sens musical. Si le prototype et les occurrences ont en commun le sous-segment final, leurs débuts sont tous différents quoique similaires. En particulier, les quatre débuts finissent par un sol dont la durée et le placement métrique sont variables d'une occurrence de Pat1 à l'autre. De plus, la mise en correspondance par le programme de deux sous-segments particuliers est intéressante : les descentes diatoniques avant le premier do dans le prototype et la première occurrence (fa-mib-ré et ré-do-si respectivement) sont appariées malgré une transposition locale. On peut aussi noter que, quoique similaires, les débuts du prototype et des occurrences n'ont pas tous la même longueur (dans l'ordre : 6, 6, 3 et 4, en excluant les silences).

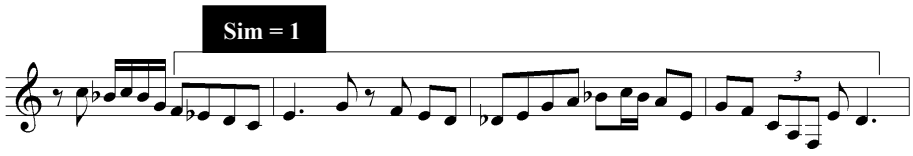


Figure 9.12. *Prototype du pattern Pat1 : passage [19:3-23:1[du solo sur Relaxin' at Camarillo – prise 4*

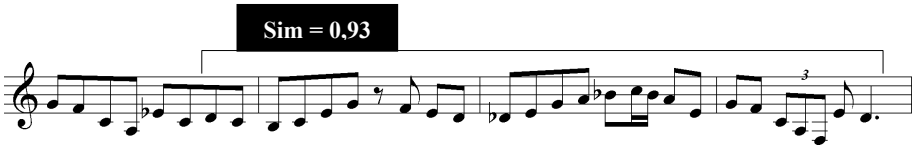


Figure 9.13. *Première occurrence de* Pat1 : *passage [7:4-11:1[du solo sur* Relaxin' at Camarillo – *prise 4*

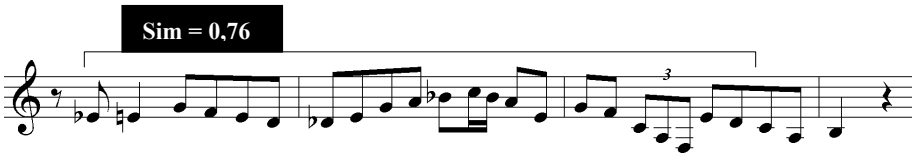


Figure 9.14. *Seconde occurrence de* Pat1 : *passage [32:1.75-35:1[du solo sur* Perhaps – *prise 3*

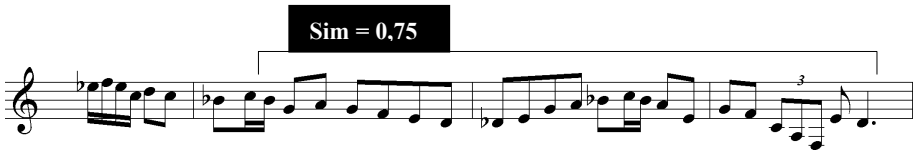


Figure 9.15. *Troisième occurrence de Pat1 : passage [20:1.5-22:4[sur Relaxin' at Camarillo – prise 3*

9.5. Segmentation

La segmentation d'une surface musicale est une tâche cruciale dans l'analyse musicale : la segmentation choisie initialement peut affecter profondément la suite de l'analyse en excluant *de facto* beaucoup de structures ne correspondant pas à des segments. Nous avons dit dans la deuxième partie de ce chapitre que la segmentation d'une surface musicale est affectée non seulement par les discontinuités locales mais aussi par des processus de plus haut niveau. Le plus important de ces processus de haut niveau est peut-être le calcul de la *similitude musicale :* les patterns musicaux similaires ont tendance à être détectés et perçus comme des unités dont le début et la fin de points influencent la segmentation de la surface musicale.

Par exemple, un certain modèle de détermination des frontières locales pourrait situer une frontière locale à l'intervalle entre la troisième et la quatrième note de *Frère Jacques* comme sur la figure 9.6 (puisque c'est un intervalle de hauteur plus

grand que ses voisins), alors qu'il est évident qu'il y a une frontière légitime entre la quatrième et la cinquième note du fait de la répétition mélodique.

Les procédures de segmentation basées sur la similitude sont très rares (voir [BAK 89, MAR 01]. Dans cette section, nous présentons une procédure basée sur l'algorithme d'induction de patterns décrit au paragraphe 9.4.1.

L'approche basée sur l'algorithme d'appariement exact que nous avons mentionné et sur la fonction de choix permet de découvrir des patterns mélodiques « significatifs », mais il faut encore appliquer un nouveau traitement qui produira une « bonne » description de la surface (en termes d'exhaustivité, d'économie, de simplicité, etc.). Quelques-uns des patterns de hauteur devront sans doute être éliminés, et il reste possible qu'une combinaison des patterns ayant un score non maximal donne une meilleure description de la surface musicale (par exemple, dans la figure 9.6, on peut se demander s'il faut préférer, comme pattern de hauteurs, *a*, *b,* ou une combinaison des deux comme *a*-*a*-*b*-*b*).

Pour résoudre ce problème, on a inventé une méthodologie très simple à utiliser, mais fruste, consistant à appliquer la procédure d'induction de patterns à autant de chaînes paramétriques que nécessaire de la surface mélodique et de ses réductions. Aucun pattern n'est ignoré ; chacun contribue à chaque frontière possible de la séquence mélodique proportionnellement à son score selon la fonction de choix. Pour chaque point de la surface mélodique, on ajoute le score de choix de tous les patterns dont un des arcs le touche, de façon à créer un profil de score de frontières de patterns, normalisé à l'intervalle [0,1]. On suppose que les points de la surface auxquels correspondent des maximums locaux sont plus susceptibles d'être perçus comme des frontières à cause de la similitude musicale. Dans l'exemple mélodique de la figure 9.16, le profil de relief de frontières de patterns a été calculé en appliquant l'algorithme d'induction exacte de pattern aux profils paramétriques d'intervalles diatoniques, de type d'intervalles et de durée. On peut observer dans la figure les fortes limites de patterns aux points indiqués par des astérisques, où les frontières locales sont très faibles.

Les frontières découvertes par l'induction de patterns peuvent être complémentaires des frontières locales détectées avec le *MDFL* pour la définition du profil de frontière globale (PFG). Le PFG est obtenu en calculant la moyenne pondérée des profils de frontières locales (PFL) et des profils de frontières de patterns (PFP), normalisés eux aussi. Dans l'application dont il est question ici, les deux profils de force ont une importance égale. Les maximums locaux du profil de frontière globale peuvent guider la segmentation de la surface musicale.

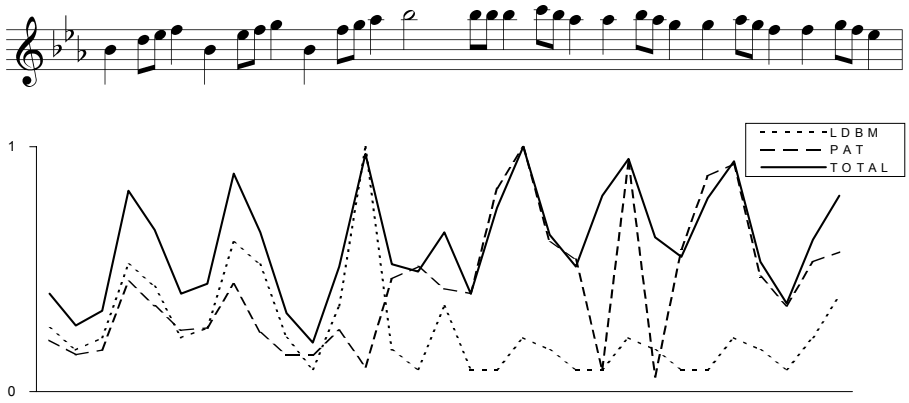


Figure 9.16. *Profil de frontières locales (PFL), profil de frontières de patterns (PFP) et profil de frontière globale (PFG) du premier thème de la* Valse opus 18 *de Chopin*

On peut choisir une ou plusieurs segmentations dans le PFG. Dans cet exemple, nous choisissons une segmentation constituée de tous les maximums locaux de valeur supérieure à 0,7 (voir la segmentation A dans la figure 9.17). Nous utilisons aussi une autre segmentation métriquement régulière basée sur le fait que la structure métrique et la structure de groupement tendent à coïncider (en phase et hors phase) ; ce groupement basé sur la coïncidence peut être déterminé en trouvant simplement la meilleure adéquation du réseau métrique précédemment découvert (3/4 dans notre exemple) sur le PFG (segmentation B, figure 9.17).

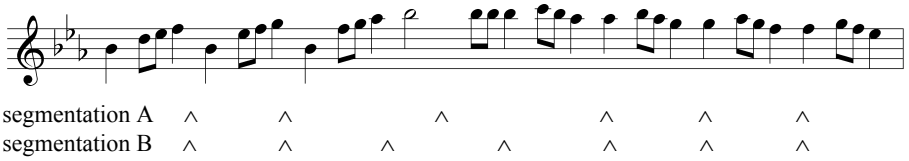


Figure 9.17. *Deux segmentations de la mélodie initiale de la* Valse *de Chopin opus 18. La segmentation A provient du PFG de la figure 9.16 en retenant les maximums locaux ≥ 0,7. La segmentation B est superposable et en phase avec la structure métrique en 3/4 de la mélodie.*

Nous allons maintenant voir comment organiser en catégories utiles (motifs, thèmes, etc.) les segments mélodiques produits lors de cette segmentation.

9.6. Similitude musicale et catégorisation

La similitude musicale semble être un phénomène difficile à décrire de façon systématique. Quand deux passages musicaux sont-ils similaires, et quand sont-ils suffisamment différents pour être considérés comme dissemblables ? Quels passages musicaux sont assez similaires pour appartenir à la même catégorie ? Comment traiter les passages ambigus ?

Höthker *et al*., [HOT 00] et Rolland [ROL 99] mentionnent quelques tentatives récentes d'application de modèles informatiques de catégorisation aux passages mélodiques. Tous les modèles informatiques de la catégorisation musicale doivent répondre aux problèmes de la détermination du nombre optimum de catégories musicales, de la détermination d'un seuil de similitude et du traitement des segments mélodiques ambigus. Nous proposons ci-dessous une façon de répondre à cette question.

Nous pensons qu'il existe une forte corrélation entre les notions de catégorisation, de similitude et de représentation des entités ou des propriétés. Les gens ne partent pas d'une description précise d'entités et de propriétés pour trouver des similarités entre eux puis des regroupements en catégories constituées selon ces similarités (figure 9.18a). Il est plus vraisemblable qu'en organisant leur connaissance du monde, ils modifient leurs représentations d'entités en même temps qu'ils créent des catégories (émergentes) et portent des jugements de similitude (figure 9.18b).

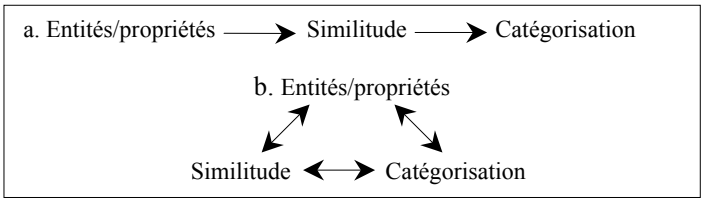


Figure 9.18. *Relations entre entités/propriétés, similitude et catégorisation*

Une de nos principales hypothèses est que la similitude dépend toujours du contexte (elle est contextuelle) et que les cas où elle semble relativement stable sont simplement les cas dans lesquels le contexte (par exemple la structure du monde ou un système culturel) est lui-même stable. Certes, il existe des contraintes perceptives générales quant à ce qui est perceptible, mais au-delà de ces contraintes, les propriétés des entités acquièrent leur prégnance dans un contexte donné, pour une certaine tâche de catégorisation ou pour certain un jugement de similitude. Tversky (1977) a montré l'importance du contexte dans les jugements de similitude et la façon dont les propriétés d'objets prennent une valeur *diagnostique* dans un contexte spécifique ; cependant il considère ces effets contextuels sur la similitude comme

des exceptions et non comme la norme, et définit la similitude indépendamment de la catégorisation.

Dans le domaine musical, fortement autoréférentiel, on peut considérer une œuvre (ou un ensemble d'œuvres) comme un contexte local comportant des motifs, des thèmes, des groupes de progression harmoniques, etc. La recherche de la similitude de deux passages musicaux isolés a généralement des résultats douteux ou peu intéressants. Par exemple, les deux patterns rythmiques 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯 & 𝅘𝅥𝅯 𝅘𝅥𝅯 sont-ils similaires ? Certaines personnes diront que oui, d'autre que non, et d'autres ne se prononceront pas. Nous pensons quant à nous que la question est mal posée et que l'on a tout lieu de ne pas se prononcer. Le problème est que ces deux patterns sont privés de contexte. Par exemple, on peut les considérer comme très différents dans un contexte musical constitué seulement de ces deux patterns (par exemple le final de la *Sonate pour violon en do majeur BWV 1005* de J.S. Bach), mais très semblables dans un contexte plus divers contenant aussi des triplets et d'autres patterns rythmiques (par exemple la première section de la *Sonate en fa majeur KV 280* de Mozart). Le contexte semble donc primordial lorsque nous faisons des jugements de ressemblance et que nous créons des catégories concernant les passages musicaux, et il n'y a pas de critère absolu servant à dire *de façon générale* quels objets sont similaires.

Sur ces principes, nous avons mis au point l'algorithme *Unscramble* [CAM 00 ; CAM 01b] qui, à partir d'un ensemble d'objets et d'un ensemble initial de propriétés, produit une série de groupes plausibles pour un contexte donné. Au cours de ce processus dynamique, l'ensemble de propriétés est ajusté pour permettre d'obtenir une description satisfaisante. Cet algorithme de regroupement détermine automatiquement un nombre approprié de groupes, choisit les attributs caractéristiques ou définitoires de chaque catégorie et permet un chevauchement limité des groupes. L'algorithme est brièvement décrit en annexe A.1. Pour une description plus détaillée, on pourra se reporter à [CAM 00].

Cet algorithme de regroupement est appliqué aux segments mélodiques illustrés dans la figure 9.17, de façon à calculer au niveau superficiel, pour chaque segment mélodique, les attributs suivants :

– divers attributs d'intervalle de hauteur, comme les patterns d'intervalles mesurés en demi-tons, les intervalles diatoniques, le type d'intervalle mélodique[4] et le contour (ascendant, *recto tono*, descendant) ;

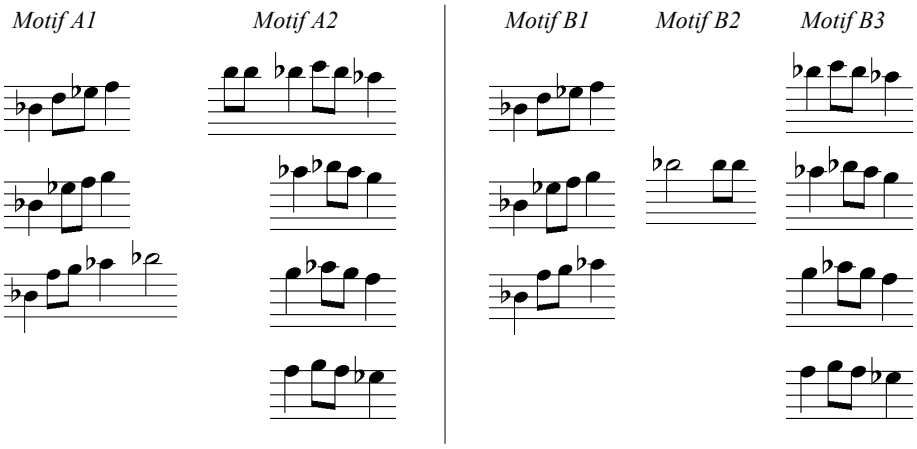
4. Type d'intervalle mélodique : (E)gal : 0 demi-ton, (C)onjoint : 1 ou 2 demi-tons, (D)isjoint : ≥ 3 demi-tons.

– divers attributs rythmiques, comme les patterns d'intervalles entre instants initiaux, les rapports de durée[5] et les comparaisons de durée de séquences (PC : plus courte, PL : plus longue, E : égale).

Par exemple, dans le premier segment de la figure 9.17a, les attributs, dans l'ordre, ont les valeurs suivantes :

– attributs d'intervalles de hauteur : 4-1-2, 2-1-1, PL-PC-PC, A-A-A ;

– attributs rythmiques : 1/4-1/8-1/8-1/4, 1/2-0-2, PC-E-PL.



a) Regroupements pour la segmentation A b) Regroupements pour la segmentation B

Figure 9.19. *Motifs produits par l'algorithme de regroupement pour les deux segmentations (figure 9.17) de la mélodie initiale de la* Valse opus 18 *de Chopin (les colonnes de segments mélodiques représentent les groupes)*

L'algorithme *Unscramble* est alors appliqué aux sept segments de la segmentation A, puis aux huit segments de la segmentation B. Les résultats du processus de regroupement sont indiqués dans la figure 9.19. Soulignons que des modèles d'attributs sont appariés s'ils sont identiques ou si l'un d'eux est un sous-modèle contigu exact de l'autre (par exemple PL-PC-PC est un sous-modèle de PL-PC-PC-PC). Non seulement l'algorithme produit des regroupements de segments mais il choisit aussi les ensembles d'attributs les plus caractéristiques de chaque groupe. Par exemple, le motif B_1 est défini par les attributs d'intervalles de degrés et de contour

5. Rapport de durée : proportion entre deux intervalles successifs, par exemple 1/2 pour une séquence noire-croche.

qui reçoivent un poids maximum de 1 tandis que les attributs rythmiques reçoivent des valeurs très faibles (tous les passages correspondant au motif B_1 ont le même pattern rythmique, mais ce pattern est aussi celui de tous les passages correspondant de B_3 ; ce pattern rythmique n'est donc pas caractéristique de B_1).

9.7.Conclusion

Dans ce chapitre, nous avons présenté un certain nombre de techniques informatiques et d'algorithmes pour essayer de décrire la structure musicale. La possibilité de mettre en œuvre des programmes capables de repérer les aspects de base de la « compréhension » musicale est cruciale, non seulement pour améliorer notre compréhension de la musique elle-même, mais aussi pour construire les outils analytiques qui peuvent être utilisés dans un grand nombre d'applications musicales pratiques, comme l'exécution interactive, la composition assistée par ordinateur, l'indexation et la recherche de données musicale, l'expressivité de l'exécution automatique, etc.

Dans ce chapitre, nous nous sommes concentrés sur des méthodes d'analyse de mélodies en termes de frontières locales, de segmentation, de patterns, de motifs, etc., mais ces techniques, étant basées sur des principes logiques et cognitifs généraux, peuvent être adaptées à d'autres sortes de tâches analytiques et concrétisées dans des programmes musicaux plus sophistiqués. Le principal objectif du chapitre était d'illustrer divers types de problèmes du traitement de la musique et quelques façons de les résoudre, plutôt que de proposer des programmes prêts à l'emploi. L'analyse musicale automatisée est un vaste territoire qui reste en grande partie inexploré.

9.8. Bibliographie

[APO 85] APOSTOLICO A., GALIL Z. (DIR.), *Combinatorial Algorithms on Words*, Série NATO ASI, Springer-Verlag, Berlin, 1985.

[BAK 89] BAKER M., « An Artificial Intelligence Approach to Musical Grouping Analysis », *Contemporary Music Review,* 3 :43-68, 1989.

[BAT 98] BATTEL G.U., FIMBIANTI R., « Aesthetic Quality of Statistic Average Music Performance in Different Expressive Intentions », *Proceedings of the XII Colloquium of Musical Informatics*, Gorizia, Italie, 1998.

[BEN 80] BENT I.D., « Analysis », dans S. Sadie (dir.), *The New Grove Dictionary of Music and Musicians*, vol. 1, Macmillan, Londres, 1980.

[BLO 85] BLOCH J., DANNENBERG R., « Real-Time Accompaniment of Polyphonic Keyboard Performance », *Proceedings of the 1985 International Computer Music Conference*, p. 279-290, 1985.

[CAM 96] CAMBOUROPOULOS E., « A Formal Theory for the Discovery of Local Boundaries in a Melodic Surface », *Proceedings of the III Journees d' Informatique Musicale*, Caen, France, 1996.

[CAM 97] CAMBOUROPOULOS E., « Musical Rhythm : Inferring Accentuation and Metrical Structure from Grouping Structure », dans M. Leman (dir.), *Music, Gestalt and Computing – Studies in Systematic and Cognitive Musicology*, Springer-Verlag, Berlin, 1997.

[CAM 98a] CAMBOUROPOULOS E., « Musical Parallelism and Melodic Segmentation », *Proceedings of the XII Colloquium of Musical Informatics*, Gorizia, Italie, 1998.

[CAM 98b] CAMBOUROPOULOS E., Towards a General Computational Theory of Musical Structure, Thèse de doctorat, Faculté de Musique et département de l'intelligence artificielle, Université d'Edimbourg, 1998.

[CAM 00] CAMBOUROPOULOS E., Widmer G., « Automated Motivic Analysis Via Melodic Clustering », *Journal of New Music Research*, 29(4) : 303-318, 2000.

[CAM 01a] CAMBOUROPOULOS E., « The *Local Boundary Detection Model (LBDM)* and its application in the study of expressive timing », *Proceedings of the International Computer Music Conference (ICMC01)*, Havana, Cuba, 2001.

[CAM 01b] CAMBOUROPOULOS E., « Melodic Cue Abstraction, Similarity and Category Formation : A Formal Model », *Music Perceptioni*, 18(3) :347-370 2001.

[CAM 01] CAMBOUROPOULOS E., CRAWFORD T., ILIOPOULOS C.S., « Pattern Processing in Melodic Sequences : Challenges, Caveats and Prospects », *Computers and the Humanities*, 35(1) :9-21, 2001.

[CLA 85] CLARKE E.F., « Structure and Expression in Rhythmic Performance », dans P. Howell *et al.* (dir.), *Musical Structure and* Cognition, Academic Press, Londres, 1985.

[CON 95] CONKLIN D., WITTEN I.H., « Multiple Viewpoint Systems for Music Prediction », *Journal of New Music Research,* 24 :51-73, 1995.

[COP 90] COPE D., « Pattern-Matching as an Engine for the Computer Simulation of Musical Style », *Proceedings of the International Computer Music Conference*, Glasgow, 1990.

[CRA 98] CRAWFORD T., ILIOPOULOS C.S., RAMAN R., « String Matching Techniques for Musical Similarity and Melodic Recognition », *Computing in Musicology*, 11 :71-100, 1998.

[CRO 81] CROCHEMORE M., « An Optimal Algorithm for Computing the Repetitions in a Word », *Information Processing Letters*, 12(5) :244-250, 1981.

[CRO 94] CROCHEMORE M., RYTTER W., *Text Algorithms*, Oxford University Press, Oxford, 1994.

[DEL 87] DELIÈGE I., « Grouping Conditions in Listening to Music : An Approach to Lerdahl and Jackendoff's Grouping Preference Rules », *Music Perception*, 4 :325-360, 1987.

[EPS 95] EPSTEIN D., *Shaping Time : Music, the Mind and Performance*, Schirmer books, New York, 1995.

[HIR 97] HIRAGA Y., « Structural Recognition of Music by Pattern Matching », *Proceedings of the International Computer Music Conference,* Thessaloniki, Grèce, 1997

[HOT 00] HÖTHKER K., HÖRNEL D., ANAGNOSTOPOULOU C., « Investigating the Influence of Representations and Algorithms in Music Classification », *Computing in the Humanities*, 35(1) :65-79, 2000.

[HOT 02] HÖTHKER K., THOM B., SPEVAK C., « Melodic Segmentation : Evaluating the Performance of Algorithms and Musical Experts », *Proceedings of the International Conference on Music and AI (ICMAI02)*, Edinbourg, (publié par Springer-Verlag, Berlin, Heidelberg), 2002.

[JAC 87] JACKENDOFF R., *Consciousness and the Computational Mind*, The MIT Press, Cambridge, 1987.

[LEE 91] LEE C.S., « The Perception of Metrical Structure : Experimental Evidence and a Model », dans P. Howell *et al.* (dir.), *Representing Musical* Structure, Academic Press, Londres, 1991.

[LER 83] LERDAHL F., JACKENDOFF R., *A generative Theory of Tonal Music*, The MIT Press, Cambridge, 1983.

[LON 82] LONGUET-HIGGINS H.C., LEE C.S., « The Perception of Musical Rhythms », *Perception*, 11 :115-128, 1982.

[MAR 01] MARSDEN A., « Representing Melodic Patterns as Networks of Elaborations », *Computers and the Humanities*, 35 :37-54, 2001.

[MEL 00] MELUCCI M., ORIO N., « A Novel Methodology for Music Information Retrieval », *Proceedings of the XIII Colloquium on Musical Informatics*, L'Aquila, Italie, 2000.

[MON 90] MONGEAU M., SANKOFF D., « Comparison of Musical Sequences », *Computer and the Humanities*, 24 :161-175, 1990.

[PAR 94] PARNCUTT R., « Template-Matching Models of Musical Pitch and Rhythm Perception », *Journal of New Music Research,* 23 :145-167, 1994.

[POV 85] POVEL D.J., ESSENS P., « Perception of Temporal Patterns », *Music Perception*, 2 :411-440, 1985.

[ROL 96] ROLLAND P.Y., GANASCIA J.G., « Automated Extraction of Prominent Motives in Jazz Solo Corpuses », *Proceedings of the 4th International Conference on Music Perception and Cognition (ICMPC'96)*, p. 491-495, Montréal, août 1996.

[ROL 99a] ROLLAND P.Y., « Discovering Patterns in Musical Sequences », *Journal of New Music Research*, 28 :4, p. 334-350, décembre 1999.

[ROL 99b] ROLLAND P.Y., GANASCIA J.G., « Musical Pattern Extraction and Similarity Assessment », dans E. Miranda (dir.), *Readings in Music and Artificial Intelligence*, Gordon & Breach, New York, 1999.

[ROL 01] ROLLAND P.Y., « FlExPat: Flexible Extraction of Sequential Patterns », *Proceedings IEEE International Conference on Data Mining (IEEE ICDM'01)*, San Jose, Californie, 29 novembre- 2 décembre, 2001.

[ROS 92] ROSENTHAL D., « Emulation of Human Rhythm Perception », *Computer Music Journal*,16(10) :64-76, 1992.

[ROW 95] ROWE R., LI T.C., « Pattern Processing in Music », *Proceedings of the Fifth Biennial Symposium for Arts and Technology*, Connecticut College, New London, 1995.

[STA 93] STAMMEN D.R., PENNYCOOK B., « Real-time Recognition of Melodic Fragments Using the Dynamic Timewarp Algorithm », *Proceedings of the International Computer Music Conference (ICMC'93)*, 1993.

[STE 77] STEEDMAN M.J., « The Perception of Musical Rhythm and Metre », *Perception,* 6 :555-569, 1977.

[TEM 01] TEMPERLEY D., *The Cognition of Musical Structures*, The MIT Press, Cambridge, 2001.

[TEN 80] TENNEY J., POLANSKY L., « Temporal Gestalt Perception in Music », *Journal of Music Theory*, 24 : 205-241, 1980.

Chapitre 10

Harmonisation musicale automatique et programmation par contraintes

10.1. De l'harmonisation à la résolution de problèmes

Les informaticiens ont longtemps considéré la musique comme un art particulièrement intéressant. L'histoire de l'informatique musicale remonte au tout début de l'histoire de l'informatique. Des programmes visant à composer de la musique, ou à « faire de la musique » à divers niveaux du processus de composition, ont été conçus depuis les années 1950, certains avec des résultats spectaculaires, comme la Suite Illiac [HILL 93] ou la symphonie de David Cope dans le style de Mozart [COP 95].

On peut remarquer que, au contraire de la musique, les arts visuels, la peinture, la danse ou l'architecture, n'ont pas reçu la même attention de la part des informaticiens. La raison que nous avançons est que, depuis les premières étapes de son évolution, la musique a été l'objet de formalisations. La musique tonale, en particulier, s'est développée dans le cadre d'une structure formelle bien adaptée à la modélisation sur ordinateur et à la résolution de problèmes. Inversement, ces formalisations ont en retour fortement influencé la composition musicale, de l'utilisation rationnelle du tempérament égal, jusqu'aux extrémismes de la musique sérielle. Cet article passe en revue les principales approches et les principaux résultats obtenus dans le domaine de l'*harmonisation automatique*, c'est-à-dire de la

---

Chapitre rédigé par François PACHET et Pierre ROY.

production d'arrangements musicaux (partitions à plusieurs voix) de mélodies données, en examinant notamment les techniques les plus utilisées pour y parvenir, à savoir, les contraintes.

10.1.1. *La musique tonale et les traités d'harmonie*

Dans cet aperçu, nous nous concentrons sur la musique tonale, non par préférence esthétique, mais parce que la musique tonale a été formalisée de façon particulièrement mathématique, par opposition à d'autres courants musicaux.

La musique tonale englobe la majeure partie de la musique occidentale produite et écoutée jusqu'à aujourd'hui : du baroque à la musique classique et romantique, au jazz, au rock et au blues. La musique tonale repose sur la notion de *tonalité*, qui est l'organisation des sons dans des groupes de notes appelés *gammes* (par exemple la gamme de Do majeur comprend les notes Do, Ré, Mi, Fa, Sol, La et Si), eux-mêmes organisés hiérarchiquement dans une œuvre musicale. On considère généralement une certaine tonalité comme la tonalité principale du morceau. On considère alors les notes de la gamme correspondante comme plus importantes que les notes qui n'en font pas partie. L'art de la musique tonale consiste précisément dans l'arrangement de notes de façon à suggérer, préparer, contourner, ou affirmer les centres tonaux. C'est l'organisation des notes dans les tonalités apparentées qui rend cette interaction possible et significative. La musique tonale est généralement opposée – au moins d'un point de vue musicologique – à la « musique atonale », comme *la musique sérielle*, inventée par Schoenberg au début du siècle, dans laquelle les douze notes (par exemple Do, Do#, Ré, Ré#, Mi, Fa, Fa#, Sol, Sol#, La, La#, Si) du système chromatique occidental ont une égale importance. D'autres sortes de musique non tonale existent, comme la musique minimaliste (représentée par des compositeurs comme John Cage ou Steve Reich), la musique ethnique traditionnelle en général (par exemple la musique balinaise ou la musique japonaise), ou la musique dite « modale », bien que l'on puisse la considérer comme une première étape de la musique tonale (par exemple le chant grégorien, ou diverses variétés du rock et du jazz).

De nombreux traités de musique tonale ont été écrits depuis des siècles. Un des traités les plus influents est probablement celui de Johann J. Fux, le *Gradus ad Parnassum*, écrit en 1725 [FUX 65], qui a été le premier à proposer des règles précises de contrepoint, c'est-à-dire l'art de combiner différentes mélodies (on pense que Haydn, Mozart et Beethoven ont étudié la composition au moyen de ce traité). La formalisation concrète de la tonalité, avec en particulier la notion de *basse fondamentale,* a été conçue par Jean-Philippe Rameau et développée dans ses célèbres traités de 1722 [RAM 85]. A l'autre extrême, le traité d'harmonie d'Arnold Schoenberg [SCH 93] est l'un des les plus aboutis de musique tonale (bien qu'il

inclue aussi des descriptions de sa théorie atonale ultérieure). Plus de mille traités d'harmonie ont été écrits, pratiquement tous les compositeurs de musique tonale ayant rédigé leur propre traité. Aujourd'hui, on considère généralement que l'évolution de la musique tonale – au moins d'un point de vue harmonique – est maintenant terminée.

La plupart des traités d'harmonie sont organisés sous la forme d'un recueil de règles indiquant la façon dont les notes peuvent ou ne peuvent être disposées. En particulier, certaines règles spécifient les relations à satisfaire pour des notes simultanées (accords) ou les séquences d'accords. Nous décrivons ces règles au paragraphe suivant.

Plusieurs remarques s'imposent ici. Premièrement, tous les traités d'harmonie ne contiennent pas exactement les mêmes règles. En fait, les règles se sont développées en même temps que la musique. Cependant, on considère qu'il existe un ensemble de règles de base valables au cours de l'évolution de musique tonale et donc toujours présentes, comme la règle des quintes parallèles, illustrée ci-dessous. Deuxièmement, ces règles ne sauraient constituer un algorithme complètement déterminé pour composer de la musique. Ce sont principalement des spécifications de combinaisons interdites de notes, ces combinaisons étant considérées comme dissonantes ou peu agréables à l'oreille. Bien sûr, le compositeur a toujours une grande latitude dans le choix des notes. Quelques compositeurs affirment même – c'est une idée générale dans l'histoire de l'art – que la liberté de la composition vient précisément des contraintes imposées par les règles. En dernier lieu, le respect des règles ne garantit pas que le résultat sera musicalement significatif ou intéressant. Par analogie avec le langage, on peut considérer les règles harmoniques comme purement syntaxiques.

10.1.2. *Les règles d'harmonie*

Nous décrivons ici quelques exemples typiques des règles d'harmonie. Nous prendrons le cas de la musique à quatre voix, c'est-à-dire constituée de quatre mélodies ou *voix* simultanées, traditionnellement interprétées par quatre chanteurs : soprano (la voix la plus aiguë), alto, ténor et basse (la voix la plus grave). Les règles d'harmonie peuvent être classées selon ce à quoi elles s'appliquent dans la partition. Nous n'avons pas l'intention de faire une liste complète de ces règles (ce qui équivaudrait à un traité d'harmonie de plus), mais plutôt de présenter des exemples caractéristiques des règles pour donner une idée de l'ensemble du problème.

Les règles les plus simples énoncent simplement que chaque mélodie a une étendue donnée (tessiture), qui correspond à l'étendue de la voix du chanteur correspondant. Par exemple, le soprano couvre deux octaves du *Do 1* (le Do du

milieu du piano) au *Do 3* (c'est-à-dire vingt-cinq notes). Une autre règle énonce que les voix ne doivent jamais se croiser.

Les règles horizontales s'appliquent aux notes successives d'une voix donnée. Par exemple, on interdit plusieurs intervalles entre les notes, comme l'intervalle de *triton* (figure 10.1) et de façon générale tous les intervalles considérés comme dissonants dans le style de la composition.

Figure 10.1. *Les deux notes entourées violent la règle des intervalles horizontaux interdits (ici le triton ascendant entre* Mi *et* Si bémol*). Les deux autres intervalles (tierce majeure ascendante et tierce mineure descendante) sont licites*

D'autres règles (règles verticales) concernent les notes simultanées ou accords. Par exemple, l'harmonisation telle qu'elle était pratiquée au début de la période baroque n'autorise que trois hauteurs différentes dans chaque accord, et oblige donc à dupliquer une des hauteurs (en général sur une octave différente).

De même, pour certains types d'accord, on interdit la duplication de certaines notes. Par exemple, pour les accords dits de « sixte » (notés « 6 »), la basse ne peut pas être répétée (figure 10.2).

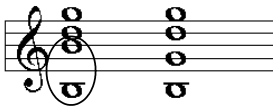


Figure 10.2. *Les deux notes entourées du premier accord violent la règle qui interdit la duplication de la basse d'un accord de sixte. L'accord suivant (qui est également un accord de sixte) est licite*

Enfin, les règles les plus complexes concernent les séquences d'accords. Une règle typique de cette catégorie est la règle des *quintes parallèles* qui interdit les quintes parallèles entre deux accords successifs. Plus précisément, si deux notes, par exemple la mélodie et la basse, d'un accord forment un intervalle de quinte, la mélodie et la basse de l'accord suivant ne doivent pas former de quinte. Cette règle est illustrée par la figure 10.3. La même règle s'applique aux octaves et aux unissons.

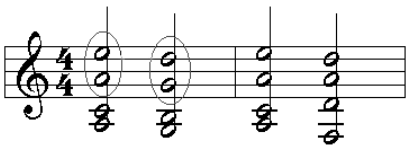


Figure 10.3. *Les deux premiers accords violent la règle des* quintes parallèles : *les deux intervalles de quinte sont entourés. Les deux derniers accords satisfont la règle des quintes parallèles*

Enfin, un autre élément important de l'harmonisation est ce que l'on appelle le chiffrage des accords. Pour poursuivre l'analogie avec le langage, le chiffrage représente une information sémantique sur l'harmonie. Depuis la musique baroque, diverses sortes de symboles ont été utilisées pour caractériser ou représenter des informations supplémentaires sur les accords dans la partition. Ces symboles (chiffrages) ont été initialement utilisés comme abréviation des accords : au lieu d'écrire les quatre notes réelles composant un accord, on n'indique que la basse et un chiffrage. Le chiffrage permet en principe à l'interprète de reconstruire l'accord d'origine ou un accord similaire. Divers systèmes de chiffrage ont été mis au point au cours de l'évolution de la musique. Il existe aujourd'hui deux grands systèmes. Le premier est le système baroque, dans lequel les chiffrages indiquent les intervalles composant l'accord : par exemple, « 6/3 » signifie que l'accord est composé d'un intervalle de sixte et d'un intervalle de tierce à partir de la note de basse. L'autre système, utilisé pour l'analyse et dans le jazz, est la notation fonctionnelle : les chiffrages indiquent la fonction harmonique de l'accord par rapport à une tonalité donnée, par exemple, l'accord de deuxième degré de la tonalité considérée. Ces systèmes, bien que très différents quant à leur rôle et à leur utilisation, véhiculent tous les deux des informations sémantiques sur l'accord. En pratique, ils donnent des informations supplémentaires permettant de trouver les notes composant l'accord. Dans la plupart des cas, il n'existe qu'un nombre restreint de possibilités pour *comprendre* un accord (c'est-à-dire trouver ses notes réelles) en fonction d'un chiffrage donné.

Comme nous l'avons dit, il n'existe aucun ensemble officiel de règles s'appliquant à tous les styles musicaux. Il y a cependant un consensus implicite quant à certaines règles, que l'on considère comme applicable à une grande variété de styles musicaux (par exemple la règle des quintes parallèles, valable du baroque au classique,). D'autres règles sont plus spécifiquement associées à un style (par exemple la limitation des accords à trois hauteurs de notes s'applique typiquement à la musique baroque, alors que la musique des époques suivantes utilise couramment des accords plus complexes). Des ensembles de règles complets sont donnés dans un format naturel dans tous les traités mentionnés ci-dessus. Un jeu de règles plus concis (vingt

règles) est donné dans Tsang et Aitken [TSA 91]. Ballesta [BAL 98] indique une liste de toutes les règles de la musique baroque sous forme de contraintes. Dans tous les cas, il faut souligner que les règles peuvent être exprimées précisément et localement, c'est-à-dire comme des prédicats (généralement des propriétés négatives) portant sur des groupes restreints de notes voisines (verticalement ou horizontalement) et leurs intervalles.

10.1.3. *La composition automatique*

Une des premières applications des règles d'harmonie dans l'informatique musicale a été la composition. Les règles harmoniques y sont compilées dans des automates à états finis. Les transitions correspondent aux suites licites d'un état donné. Si des probabilités sont associées à chaque transition, on obtient un modèle de Markov, dont les premiers utilisateurs ont été Hiller et Isaacson [HILL 93] dans la célèbre *Suite Illiac*, entièrement composée selon ce processus en 1958. Dans cette approche, les règles sont utilisées implicitement comme des contrôles passifs. Le système applique une procédure simpliste de génération et de test, sans effectuer de recherche combinatoire.

10.1.4. *Le problème de la combinatoire*

Le problème de la combinatoire apparaît quand il s'agit de produire des harmonisations avec une voix imposée. Dans ce cas, le problème de l'harmonisation devient naturellement un problème de satisfaction de contraintes (CSP) sur un domaine fini : on peut considérer chaque note inconnue comme une *variable* du CSP dont le domaine est l'ensemble des notes de la gamme de la mélodie. Ce domaine pouvant être restreint aux seules notes appartenant à la tessiture de la voix considérée. On peut alors considérer les règles harmoniques comme des *contraintes* sur ces variables, dans le sens de la satisfaction de contrainte, c'est-à-dire des relations qui doivent être satisfaites dans toutes les solutions (une solution à un CSP est, par définition, une instanciation de toutes les variables du CSP qui satisfait toutes les contraintes du CSP).

Plusieurs variantes de ce problème de base existent dans la pratique des musicologues, selon ce qui est imposé. Parmi les exercices traditionnels des classes de composition figurent les suivants :

– *chant donné* : la voix de soprano seule est connue, résoudre l'exercice consiste à composer les trois voix inférieures (alto, ténor et basse) ;

– *basse non chiffrée* : la basse étant imposée, il faut composer les trois voix supérieures (mélodie, alto et ténor). Dans ce cas, il y a seulement une quantité minimale d'informations donnée au départ. Le problème de la basse non chiffrée a

été longtemps débattu dans la théorie de musique, et divers travaux ont montré qu'il est, en pratique, très sous-déterminé (voir la thèse de doctorat de Rothgeb [ROT 68]). La figure 10.4 montre une solution d'un problème de basse non chiffrée à quatre voix ;

– *basse chiffrée* : la voix de basse est imposée, ainsi que des chiffrages (figurant au-dessus de chaque note de la basse) indiquant la nature de chaque accord. Cette information réduit considérablement les accords possibles, et de ce fait, elle réduit la combinatoire du CSP correspondant. Il a correspondu à une pratique des clavecinistes qui devaient jouer – en temps réel – des accords satisfaisant la voix de basse, les chiffrages et une mélodie chantée ;

– *problèmes à deux voix* : une mélodie étant imposée, il s'agit de trouver la basse seulement. Dans ce cas, on n'applique qu'un sous-ensemble des règles – ou des règles simplifiées.

Il convient de souligner que les deux premiers problèmes introduisent naturellement un point de vue combinatoire. Dans les deux autres, du fait de l'information supplémentaire qui est fournie, l'aspect combinatoire est moins important. L'aspect intentionnel de la composition et l'aspect combinatoire sont plus ou moins exclusifs. Considérons par exemple la figure 10.3 : une solution correcte (les deux derniers accords) produit une progression tonale tout à fait différente (le dernier accord est un accord de Ré mineur) de la première solution, qui est une progression incorrecte (le deuxième accord est un accord de Sol majeur).

Cependant, le problème strictement combinatoire est intéressant en tant que tel du point de vue des contraintes. La section suivante aborde les principales approches permettant de résoudre ce problème à l'aide de contraintes.

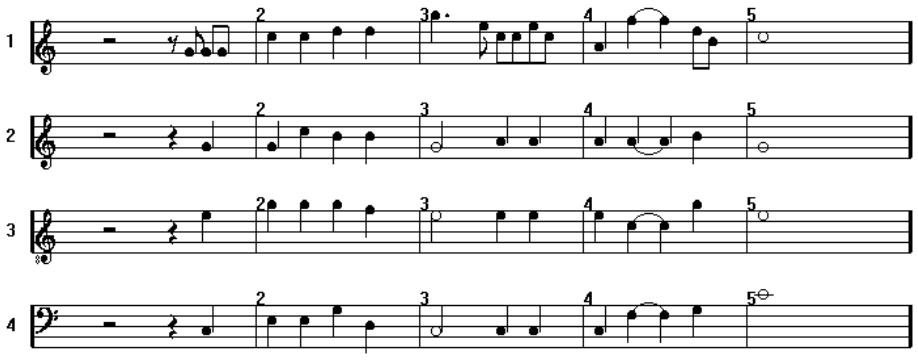


Figure 10.4. *Une harmonisation à quatre voix de style baroque. La première voix (soprano) est imposée. Les trois autres voix (2 à 4) doivent être composées pour satisfaire les règles d'harmonie baroque (d'après [ROY 98])*

10.2. Harmonisation automatique sous contraintes

Cette section passe en revue les principaux travaux essayant de résoudre un des problèmes de l'harmonisation automatiques, notamment les travaux fondés sur la notion de contraintes.

10.2.1. *Les premières tentatives de modélisation du problème par les contraintes*

Les premiers travaux étaient des tentatives innovatrices utilisant une approche déclarative pour représenter les règles musicales. Aucun algorithme de satisfaction de contraintes n'était utilisé, le souci principal étant de contrôler l'explosion combinatoire en fournissant au résolveur davantage de connaissances.

La première tentative de représentation de règles musicales sous forme de contraintes, c'est-à-dire comme des relations déclaratives portant sur des représentations musicales, est probablement celle de Steels [STE 79]. Dans ces travaux, Steels propose d'utiliser des contraintes pour créer des accords de passage, c'est-à-dire les accords qui peuvent être insérés entre deux accords donnés. Ces accords de passage doivent satisfaire diverses contraintes musicales, par exemple des relations d'intervalle entre les fondamentales de l'accord initial, de l'accord de passage et du dernier accord. Steels utilise essentiellement un système de *frames* associé à une recherche en largeur d'abord. L'explosion combinatoire n'est pas explicitement traitée, à l'exception d'un mécanisme d'évaluation « paresseux » intelligent.

Le même auteur a expérimenté plus tard un système musical plus perfectionné visant à résoudre le cas général du problème de l'harmonisation à quatre voix [STE 86]. Ce système possède la propriété intéressante de combiner l'exploration « brutale » (sans procédure de consistance) et la recherche heuristique. Un de ses buts est en fait d'apprendre automatiquement les heuristiques à partir des solutions.

Courtot [COU 90] décrit un système écrit en Prolog-II et utilisé pour définir des structures musicales et les contraintes qui les relient, afin de créer des polyphonies. Ce système, appelé Clara, vise à représenter les modèles du compositeur, et il est fondé sur un système extensible de types. Bien que la notion de contrainte soit explicitement introduite pour définir des types, l'accent est davantage mis sur l'induction de types que sur l'exploration de l'espace de recherche.

Daniel Levitt décrit dans sa thèse [LEV 81, LEV 93] un langage musical à contraintes servant à définir les propriétés stylistiques des accompagnements harmoniques. Ce langage permet de spécifier incrémentalement des contraintes, en commençant par de simples tessitures de voix, pour arriver au mouvement des fondamentales d'accords. La syntaxe de ce langage permet de spécifier des relations

arithmétiques de base entre des hauteurs et des accords, dont on prouve qu'elles suffisent à décrire des œuvres pour piano dans le style ragtime. Le résolveur utilise apparemment une procédure simple de retour arrière (*backtracking*), et n'a donc qu'une capacité restreinte de limiter l'exploration combinatoire. Par conséquent, il ne convient qu'à des problèmes de petite taille ou faciles, tels que les problèmes sous-contraints.

10.2.2. *Utilisation de la satisfaction de contraintes*

Ce paragraphe décrit les travaux d'harmonisation automatique qui utilisent la vraie satisfaction de contraintes sous diverses formes.

Schottstaedt [SCHO 89] décrit un système complet d'harmonisation à quatre voix basé sur le traité de Fux [FUX 65]. Schottstaedt fait notamment une analyse détaillée de la théorie de Fux, et en classifie les règles selon leur importance, représentée par une pénalité : interdiction (pénalité infinie), violation grave (pénalité de 200), violation bénigne (pénalité de 100) et d'autres règles assorties de pénalités plus faibles. La quinte parallèle est un exemple d'interdit (c'est-à-dire qu'aucune exception à cette règle n'est acceptée). La règle de la tessiture (c'est-à-dire les voix doivent rester dans leur tessiture) fait l'objet d'une violation grave. Le système utilise une stratégie de retour arrière de type « meilleur d'abord ».

Pour éviter de rester coincé dans un local extremum et de réduire le temps de calcul, l'algorithme abandonne la branche actuelle de l'arbre quand il trouve une solution. L'algorithme est donc incomplet, mais il est capable de trouver rapidement de bonnes solutions.

Ebcioglu [EBC 87] est probablement le premier à avoir réalisé un système capable de produire la musique à quatre voix de haute qualité, de façon entièrement automatique. Son système, qui est le fruit d'un travail considérable, contient divers modules interconnectés traitant différents aspects du processus de composition, dont l'harmonisation, la génération de mélodies, l'analyse de mélodies et la génération de notes de passage. Sur le problème de l'harmonisation lui-même, le système d'Ebcioglu contient environ 350 règles, représentant le style de Johann Sebastian Bach tel qu'analysé par Ebcioglu. Ces règles sont représentées dans un langage à contraintes spécifiquement conçu pour cette tâche, appelée BSL. BSL est essentiellement un langage de programmation logique par contraintes mettant en œuvre le saut arrière (*backjumping)*, technique que l'auteur considère nécessaire dans la recherche combinatoire spécifique à l'harmonisation musicale.

Le système d'Ebcioglu est intéressant parce qu'il est le premier système capable de résoudre un problème musical difficile et parce qu'il produit des résultats de haute qualité. Cependant, il est difficile à comprendre et à évaluer, car il est réalisé dans un langage spécial et contient de très nombreux modules interdépendants. Dellacherie [DEL 98] a essayé de clarifier ce système du point de vue du traitement de contraintes en extrayant son système de règles et en le réécrivant en Eclipse ; les résultats ont été médiocres, en raison du manque d'expressivité d'Eclipse (c'est un langage à contraintes qui ne traite pas complètement la cohérence d'arcs, inventé par Mackworth [MAC 77], dans des domaines finis). Comme l'a montré Ovans, le traitement de la cohérence d'arcs est nécessaire dans l'harmonisation musicale ; la cohérence partielle (*bound consistency*) n'est pas suffisante pour obtenir des performances acceptables.

Ovans et Davidson [OVA 92b] (voir aussi [OVA 92a]) ont été les premiers à souligner l'aspect combinatoire profond de l'harmonisation entièrement automatique et à préconiser l'utilisation de la cohérence d'arcs. Ils ont réalisé un système destiné à résoudre l'harmonisation à deux voix selon les règles de Fux, et ont comparé sur le même problème, plusieurs techniques de résolution, comme le retour arrière classique, la vérification avant (*forward checking*) et la cohérence d'arcs. Bien qu'aucune méthode ne semble être toujours meilleure que les autres, Owens souligne que la consistance d'arcs devrait toujours être utilisée pour résoudre les problèmes d'harmonisation, et critique sur ce point le système d'Ebcioglu, qui aurait été, selon lui, nettement plus efficace s'il avait eu recours à la consistance d'arcs plutôt qu'au saut arrière. Il est intéressant de noter que l'on considère aujourd'hui ces conclusions comme valides dans la programmation par contraintes en général. Cependant, le système d'Ovans ne produisant que des harmonies à deux voix, ses résultats ne sont pas entièrement convaincants.

Le premier système d'harmonisation automatique à quatre voix, à l'aide de techniques de programmation logique avec contraintes, est décrit dans Tsang et Aitken [TSA 91]. Ce système effectue une harmonisation à quatre voix à l'aide d'un jeu de vingt règles ; il est mis en œuvre dans CLP(R) [JAF 87]. Ces auteurs utilisent une représentation directe des objets musicaux (hauteurs, intervalles et types d'accords). Le résultat est un système très coûteux en temps et en volume (70 mégaoctets de mémoire pour harmoniser une mélodie à onze notes). Cependant, du fait de la simplicité du jeu de règles, il est souvent utilisé comme terme de comparaison.

Une étude très détaillée de l'harmonisation à quatre voix utilisant des techniques de satisfaction de contrainte conventionnelles est la thèse de doctorat de Philippe Ballesta [BAL 98]. Ballesta a transcrit un jeu complet de règles d'harmonisation à l'aide du système *PECOS* d'Ilog, ancêtre d'Ilog Solver. Le problème ainsi résolu est celui de la *basse chiffrée*, et ce travail inclut des descriptions détaillées de la

représentation des objets musicaux et des contraintes. Les solutions produites par ce système sont généralement considérées comme relativement bonnes du point de vue musical. La performance du système, cependant, n'est pas aussi satisfaisante, en particulier parce qu'il utilise des contraintes pour représenter toutes les relations musicales, au-delà des règles harmoniques. Par exemple, la relation entre un accord et sa note de basse est représentée comme une contrainte, au même titre que la relation entre deux notes et l'intervalle entre elles, ou bien encore les relations entre la hauteur d'une note et son nom dans une gamme donnée. Ce choix radical produit un système élégant et expressif, mais limité dans sa capacité de manipulation de mélodies très complexes ou très longues.

10.2.3. *Structuration du problème*

Les diverses tentatives de résolution du problème de l'harmonisation avec les techniques de contraintes *stricto sensu* décrites ci-dessus ont montré que les contraintes sont un paradigme utile, mais qu'elles sont toujours limitées et, dans l'ensemble, incapables de traiter des mélodies réalistes en un temps raisonnable. D'autres approches ont été utilisées pour essayer d'améliorer ces travaux, notamment en cherchant les meilleures représentations possibles du problème. Nous décrivons ici deux de ces approches.

La première consiste dans l'exploitation plus active des structures d'accords. Dans les approches précédentes, les accords étaient toujours traités comme des groupes simples de notes (ce qu'ils sont, d'ailleurs, en un certain sens). Cependant, du point de vue des contraintes, c'est une erreur. En effet, certaines règles harmoniques importantes (comme la règle des quintes parallèles décrite au paragraphe 10.1.2) manipulent explicitement des accords et non des notes. Bien sûr, il est possible de représenter ces règles comme des contraintes entre notes. C'est notamment l'approche suivie par Ovans, Ballesta ou Tsang et Aitken.

Nous avons montré dans [PAC 95] qu'en ajoutant à la définition du CSP des variables représentant les accords (en plus des variables représentant les notes de base), on pourrait réduire considérablement la complexité théorique du problème. Plus précisément, la complexité peut être réduite d'un facteur équivalent à la densité d'accords dans l'espace des quadruplets de notes. *A priori*, l'introduction de variables (au sens des CSP à domaines finis) représentant les accords pose un problème combinatoire élémentaire : le domaine de telles variables doit contenir tous les accords de quatre notes possibles dans une tonalité donnée, et ces accords sont en bien trop grand nombre pour que le problème soit possible à résoudre de façon efficace (la taille des domaines des variable est un paramètre essentiel de la combinatoire d'un CSP). On contourne cette difficulté en décomposant le processus de satisfaction de contraintes en plusieurs phases.

Dans un premier temps, on ne considère que les variables représentant les notes et les contraintes (ou règles musicales) entre ces « variables notes », de façon à réduire l'ensemble des valeurs de notes possibles pour chaque « variable note » du problème. A l'issue de cette première phase, les domaines des variables notes sont sensiblement réduits, ce qui permet de définir des variables représentant les accords dont les domaines sont de taille « raisonnable » (techniquement, le domaine d'une « variable accord » est équivalent au produit cartésien des domaines des quatre « variables notes » le composant. De ce fait, plus ces variables notes ont des petits domaines, plus le nombre d'accords que l'on peut construire est petit) ; enfin, une dernière phase consiste à créer et résoudre le CSP contenant les « variables accords » et les contraintes d'accords (c'est-à-dire, les contraintes correspondant à des règles musicales entre accords, comme la règle des quintes parallèles déjà présentée à plusieurs reprises dans cet article). Les solutions de ce problème sont les harmonisations recherchées.

En pratique, cette approche permet d'obtenir quasiment en temps réel les solutions d'exercices à quatre voix (avec ou sans chiffrage). Les détails de la conception et de la mise en œuvre sont fournis dans [ROY 98]. Soulignons que ces résultats sont obtenus sans ajout de connaissances au système.

Une autre approche a été proposée pour réduire la complexité du problème d'harmonisation [HEN 96, ZIM 01]. Elle consiste à construire, avant la résolution, un plan d'harmonisation contenant les informations sur l'harmonie à produire. Ce plan représente l'intention harmonique du morceau, qui avait disparu lors de l'introduction du problème combinatoire (voir paragraphe 10.1.4). Le plan contient des informations de chiffrage comme le degré (par exemple I ou II ; voir paragraphe 10.1.2). Cette information réduit considérablement le nombre de notes qu'il est possible d'utiliser, puisqu'il détermine les classes de hauteur de chaque accord (par exemple Do, Mi, Sol) et ne laisse indéterminées que les octaves réelles de ces classes de hauteur (par exemple. Do0, Mi2, Sol2, Do2). Une fois que le plan harmonique est produit, un système de contraintes directes (écrit en Oz) calcule les notes réelles (ce que l'on appelle la « réalisation » des accords). Ce second problème est beaucoup plus simple que le problème de l'harmonisation d'un chant donné (donc sans chiffrage) à quatre voix.

10.2.4. *Eléments algorithmiques*

Il serait long et fastidieux de décrire en détail l'algorithme complet d'un système de résolution de contraintes pour l'harmonie. Néanmoins, on peut facilement esquisser à grands traits son fonctionnement.

L'algorithme de résolution de la plupart des résolveurs de contraintes comme le système BackTalk est fondé sur le principe de la *cohérence d'arcs*, définie par Mackworth [MAC 77]. Une contrainte C, portant sur deux variables X et Y, est dite arc cohérente si pour toute valeur x du domaine de X, il existe au moins une valeur y dans le domaine de Y telle que C(x,y) (c'est-à-dire, telle que le couple de valeur (x,y) satisfasse la contrainte C). Cette définition se généralise de façon évidente aux contraintes portant sur plus de deux variables.

Une remarque importante est que l'on peut toujours rendre arc cohérente une contrainte en supprimant des valeurs dans les domaines des variables impliquées. Pour s'en persuader, il suffit de penser que si l'on supprime toutes les valeurs des domaines des variables, la contrainte est alors trivialement arc cohérente.

Une propriété fondamentale des CSP est que l'on peut toujours rendre une contrainte arc cohérente en réduisant les domaines des variables *sans modifier l'espace des solutions* du CSP. On peut ainsi utiliser la cohérence d'arcs pour réduire la taille du CSP, et donc sa combinatoire, tout en ne supprimant aucune solution.

Voici un algorithme simple pour le faire : soient deux variables X et Y de domaines dom(X) et dom(Y), et C une contrainte portant sur X et Y :

Algorithme A :

```
pour tout x dans dom(X)
  si pour tout y dans dom(Y) on a non(C(x,y))
  alors supprimer la valeur x de dom(X)
pour tout y dans dom(Y)
  si pour tout x dans dom(X) on a non(C(x,y))
  alors supprimer la valeur y de dom(Y)
```

Il est aisé de vérifier que l'application de cet algorithme, à une contrainte quelconque du CSP, ne modifie aucunement l'espace des solutions, puisque les valeurs que l'on supprime des domaines ne peuvent faire parties d'aucune solution.

L'algorithme sur lequel s'appuient les résolveurs de contraintes est le suivant : soit P un CSP, nous notons dom(V) le domaine d'une variable V :

Algorithme B :

```
tant qu'il existe C non arc cohérente P
rendre C arc cohérente en utilisant l'algorithme A
```

Dans BackTalk, l'algorithme B est inclus dans une boucle d'exploration arborescente dont voici une description simplifiée.

Mécanisme général de résolution des CSP dans BackTalk :

```
1. tant qu'il existe dans P une variable non instanciée
2.      soit v une telle variable
3.             choisir x dans dom(v)
4.                    empiler l'état courant de P
5.                    instancier v avec la valeur x
6.                    rendre P arc cohérente avec
Algorithme B
7.                    si
8.                           il existe v tel que dom(v)={}
9.                    alors
10.                          dépiler l'état précédent de P
11.                          supprimer x de dom(v)
12. si toutes les variables sont instanciées
13.     retourner la solution
14. sinon
15.     il n'existe pas de solution
```

La ligne 5 consiste à explorer une nouvelle branche de l'arbre de recherche. Les lignes 10 et 11 consistent à abandonner l'exploration de la branche choisie en ligne 5, et à continuer dans le reste de l'arbre de recherche. A chaque nouvelle branche, on applique l'algorithme B afin de réduire la taille du problème sans changer l'espace des solutions.

10.2.5. *Les techniques de recherche locale*

Une autre approche de la satisfaction de contraintes consiste à exploiter des algorithmes incomplets, mais qui sont plus adaptés aux problèmes pour lesquels une solution approchée suffit, et qui ont relativement une grande densité de solutions. En composition notamment, des solutions approchées sont souvent aussi satisfaisantes que d'hypothétiques solutions exactes. Le système OMClouds de Charlotte Truchet [TRU 03], est une bibliothèque de contraintes intégrée au langage visuel OpenMusic (voir le chapitre sur la programmation visuelle pour la composition, même volume). OMClouds s'appuie sur un algorithme incomplet de recherche locale qui fournit très rapidement des solutions approchées, c'est-à-dire violant aussi peu de contraintes que possible.

Techniquement, ces algorithmes incomplets sont basés sur un mécanisme de coûts : chaque contrainte est associée à un coût en cas de violation, ce qui permet de pondérer les différentes contraintes du problème selon leur importance. En outre, OMClouds reprend la philosophie du système *Situation* concernant les variables : elles ne représentent pas directement des objets musicaux, mais des nombres entiers, l'association entre ces nombres et les objets musicaux est de la responsabilité des utilisateurs. L'intégration de OMClouds dans OpenMusic en fait un outil destiné aux compositeurs. OMClouds a été utilisé, par exemple, pour classer une suite

d'accords de façon à maximiser le nombre de notes communes entre deux accords successifs ; ou encore pour résoudre le problème d'harmonisation automatique de canons rythmiques dans *Empreinte sonore de la fondation Bayeler* de Georges Bloch.

Le OMClouds repose sur l'algorithme suivant, dû à Codognet et Diaz [COD 01] : on parcourt l'espace de recherche en se guidant par une mesure de la qualité de l'instanciation courante des variables, et en évitant de revenir trop souvent aux mêmes instanciations. Cela suppose une mesure de la qualité des instanciations, qui est donnée par une fonction de coût sur les contraintes.

L'algorithme utilise ensuite les projections des coûts sur chaque variable du problème afin de trouver la variable la « plus coûteuse » et d'essayer d'améliorer l'instanciation courante en changeant la valeur de cette variable. Voici l'algorithme :

E est le seuil d'arrêt de l'algorithme, normalement fixé à 0, Ltabu est la longueur de la liste tabou, et V+ la plus mauvaise variable pour une itération donnée :

```
Initialisation aléatoire des variables
Répéter
   1.Calcul des coûts de toutes les variables sauf les
variables marquées tabou, sélection de la plus chère V+
   2.Test du coût global en remplaçant V+ par toutes les
valeurs de son domaine, sélection de la valeur la plus
avantageuse v'
   3.Si à la fin de l'exploration, aucune valeur
n'améliore le coût global, alors V+ est marquée tabou
pendant Ltabu itérations, sinon, V+ est instanciée à v'
   4.Si toutes les variables sont tabou, alors
réinitialisation aléatoire
Jusqu'à avoir une erreur globale inférieure à E
```

Un des points remarquables de cet algorithme est qu'il est très rapide, en comparaison des algorithmes complets. Par ailleurs, au contraire des algorithmes de recherche complets, il peut être arrêté à tout instant, pour récupérer la meilleure solution trouvée jusqu'alors.

Notons enfin qu'il existe d'autres applications de cet algorithme à la musique, par exemple pour le séquencement temporel de samples audio, avec la technique du *musaicing* introduite par Zils et Pachet [ZIL 01], ou bien encore la génération de playlists pour les systèmes de gestion de contenu musicaux : voir [PAC 00] pour des techniques complètes, et [AUC 02] pour des techniques à base de recherche locales, similaires à celle écrite ici.

10.2.6. *Autres approches*

Une série de résolveurs de contraintes conçus pour des problèmes musicaux a été réalisée à l'IRCAM et utilisée par les compositeurs de musique contemporaine. L'un des plus récents, *Situation*, permet de spécifier des polyphonies à l'aide d'un jeu de contraintes spécifiques intégrées [RUE 98]. Il n'est pas destiné à des exercices de résolution de musique tonale, mais est plutôt conçu comme outil concret de composition. Une caractéristique intéressante du moteur est sa capacité à énoncer les contraintes de façon abstraite : elles peuvent ainsi porter sur des « points » (par exemple des notes, des accords ou des durées) et sur des « distances » (par exemple des intervalles de hauteur, des intervalles de temps, des intervalles d'accords, etc.). Cette couche abstraite permet d'utiliser le résolveur dans un grand nombre de situations, et pas uniquement dans le domaine des hauteurs, car l'utilisateur peut choisir la façon d'associer ces « points » et ces « distances » aux entités musicales.

D'un point de vue technique, le moteur de Situation est basé sur un mécanisme de vérification *a priori* (*forward checking*) assorti d'une évaluation paresseuse [RUE 97] et utilise une représentation unidimensionnelle du problème (c'est-à-dire que des variables représentent des notes ou des attributs de notes). Des expériences menées pour définir avec *Situation* des contraintes correspondant aux règles de l'harmonie tonale [GUE 98] ont fait apparaître les mêmes limitations que dans les travaux de Ballesta. Cependant, les contraintes intégrées permettent de trouver des solutions rapidement et sont particulièrement bien adaptées aux besoins de compositeurs de musique contemporains.

Le travail de Ramirez et Peralta [RAMI 98], bien que non directement lié à l'harmonisation à quatre voix, mérite d'être mentionné ici : ils abordent la question de la production d'une séquence d'accords (en réalité une séquence de noms d'accords) qui harmonise de façon satisfaisante une mélodie donnée. Ils se limitent à des accords de trois notes. L'aspect intéressant de ces travaux est qu'ils essaient de maximiser le nombre de suites d'accords familières, qui sont stockées dans un dictionnaire. Un algorithme simple, qui n'est pas fondé sur les techniques de cohérence d'arcs, permet de résoudre le problème. Dans ce cas, la véritable satisfaction de contraintes n'est pas nécessaire, puisque la recherche combinatoire est très réduite par rapport au problème de l'harmonisation à quatre voix. Cependant, le système résout un véritable problème d'harmonisation musicale.

Enfin, Phon-Amnuaisuk et Wiggins [PHO 98] proposent une comparaison entre une approche à base de règle (représentation de la connaissance explicitement symbolique) et des approches fondées sur des algorithmes génétiques ; ils concluent à la supériorité de la première approche du point de vue de la qualité des solutions obtenues. Bien sûr, il reste difficile de fournir explicitement au système les règles d'harmonie. Des travaux récents ont abordé cette question en essayant d'induire

automatiquement des jeux de règle ou des grammaires à partir de divers recueils de partitions à l'aide de techniques statistiques [PON 99].

10.2.7. *Les systèmes existants*

Il n'existe pas, à notre connaissance, de logiciel commercial d'harmonisation fondé sur des techniques de contraintes, sans doute parce que bien que les problèmes d'harmonisation (notamment dans la musique tonale) soient des exercices intéressants, ils n'ont que des applications limitées. En pratique, de nombreux synthétiseurs proposent des fonctions d'harmonisation de base (comme des accords obtenus à l'aide d'une seule touche sur certains claviers électroniques). Ces harmonisations sont généralement exécutées de façon réactive sur la base d'une note associée à un accord, c'est-à-dire sans tenir compte du contexte (les notes précédentes de la mélodie). Dans ces systèmes, les exigences du temps réel interdisent de toute façon l'utilisation du retour arrière !

Le système *Tonica* (distribué par Software Partners) est un excellent système d'harmonisation à base de techniques connexionnistes. Il harmonise des mélodies qu'on lui donne, et possède une interface très flexible. Le système procède en deux phases. Il produit d'abord une analyse de la mélodie en termes d'accords (l'équivalent des chiffrages), puis une harmonisation réelle de la mélodie et la description précise des accords. Comme de nombreuses analyses harmoniques sont possibles pour une mélodie donnée, le système n'effectue pas de recherche profonde. L'utilisateur a la responsabilité de choisir une analyse qui convient à son intention. En pratique, cette approche est beaucoup plus intuitive pour des musiciens, et le système a donc une utilité certaine. Toutefois, il ne propose pas de nouvelle solution au problème de l'harmonisation du point de vue de la résolution de problèmes. Par ailleurs, plusieurs approches ont été proposées pour la résolution de problème d'harmonisation à l'aide de techniques d'apprentissage comme les réseaux connexionnistes, par exemple dans [HIL 92], avec d'excellents résultats.

10.3. Conclusion : le problème est-il résolu ?

On peut maintenant considérer le problème technique de l'harmonisation à quatre voix comme résolu grâce aux techniques de satisfaction de contraintes utilisant la cohérence d'arcs associée à une structuration adéquate du sous-problème de la manipulation correcte des variables d'accord. On en est arrivé là après plusieurs années d'essais et d'erreurs, en commençant par des approches « brutales » (par exemple, celle de Schottsdtaedt) pour passer à des langages à contraintes spécifiques (Ebcioglu) et à techniques de consistance d'arcs (Ovans), assorties d'une bonne structuration du problème (Pachet et Roy).

Cependant, une partie non résolue est le sous-problème de la production de mélodies musicalement agréables ou intéressantes. Divers aspects de ce qui rend les mélodies intéressantes demeurent non résolus.

Tout d'abord, on ne connaît pas de règles rendant compte de ce qui fait qu'une mélodie est intéressante, pour autant que de telles règles existent. Comme nous l'avons vu, la formalisation de la musique a surtout porté sur l'établissement de « règles syntaxiques ». Plusieurs tentatives ont été faites pour formuler les propriétés explicites des mélodies qui semblent « agréables » ou « équilibrées » (voir par exemple les approches statistiques et les travaux sur la propriété « 1/f » en musique [VOS 78]), mais leurs formulations n'ont pas atteint un niveau permettant de les convertir en spécifications opérationnelles.

Deuxièmement, l'intérêt musical résulte probablement de l'interaction entre des critères incompatibles. Par exemple, il est bien connu que des mouvements conjoints (c'est-à-dire des voix allant dans la même direction, vers le haut ou vers le bas) semblent agréables, mais cela va à l'encontre des diverses règles opposées aux mouvements parallèles (quintes ou octaves parallèles). Comment ces critères coexistent-ils exactement ? Les travaux de Ballesta décrits ci-dessus comportent des études des critères d'optimisation choisis parmi les diverses solutions. Ballesta propose plusieurs critères pour décrire des solutions mélodiques et les met en œuvre à l'aide de l'algorithme de séparation-évaluation (*branch and bound*) du résolveur. Par exemple, une solution agréable peut être décrite par les propriétés suivantes :

– la voix de soprano ne doit pas changer trop souvent de direction ;

– le soprano et la basse doivent être autant que possible en mouvement contraire ;

– la voix de soprano doit se déplacer autant que possible par intervalles conjoints (notes voisines) ;

– les voix intermédiaires doivent être maintenues aussi proches que possible .

Ce problème d'optimisation multicritère est très difficile et aucune expérience systématique n'a encore été faite sur des mélodies réalistes, mais la question mérite certainement davantage d'attention.

Enfin, les mélodies agréables sont des mélodies qui contiennent des modèles familiers et des suites harmoniques ou des « solutions » harmoniques typiques. Mais dans le contexte de systèmes d'harmonisation automatique, il est difficile de se conformer à cette exigence. En effet, nous sommes loin d'avoir élaboré un catalogue de formules harmoniques familières, *a fortiori* de pouvoir favoriser l'utilisation de telles formules dans les harmonisations produites. Les travaux de Ramirez et Peralta constituent un premier effort dans cette direction.

L'harmonisation automatique a depuis longtemps soulevé des questions difficiles pour le domaine de la programmation par contraintes et a incité les chercheurs à trouver des solutions à ces questions, élargissant ainsi notre connaissance des structures musicales tout comme celle des problèmes de satisfaction de contraintes en général. Comme nous avons essayé de le montrer, ce problème – ou plutôt cette catégorie de problèmes – continuera à apporter des questions stimulantes à la recherche sur les contraintes.

10.4. Bibliographie

[AUC 02] AUCOUTURIER J.J., PACHET F., « Scaling up Music Playlist Generation », *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*, Lausanne, Suisse, août 2002.

[BAL 98] BALLESTA P., « Contraintes et objets : clefs de voûte d'un outil d'aide à la composition », dans M. Chemillier et F. Pachet (dir.), *Informatique Musicale, Recherche et Applications*, , Hermès, Paris, 1998.

[BRO 94] BROWN F., MOUTON R., « L'automate musical en temps réel Kantor », *Premières Journées d'Informatique Musicale*, Labri, Bordeaux, 1994.

[COD 01] CODOGNET P., DIAZ D., « Yet another local search method for constraint solving, LNCS 2246 », *First Symposium on Stochastic Algorithms: Foundations and Applications, SAGA 2001*, Berlin, Allemagne, p. 73-90, 13-14 décembre 2001.

[COP 87] COPE D., « An Expert System for Computer-Assisted Music Composition, » *Computer Music Journal*, 11 (4), p. 30-46, 1987.

[COU 90] COURTOT F., « A Constraint Based Logic Program for Generating Polyphonies », *International Computer Music Conference,* Glasgow, 1990. Voir également dans Balaban M. *et al*. (dir.), *Understanding Music with AI*, MIT Press/AAAI Press, Cambridge, p. 156-181, 1992.

[DEL 98] DELLACHERIE P., Modélisation et optimisation d'un harmoniseur automatique de chorals en PLC (FD), Rapport de DEA, Université de Caen, 1998.

[EBC 87] EBCIOGLU K., Report on the Choral Project: An Expert System for Harmonizing Four-Part Chorales, IBM Rapport technique RC 12628, 1987.

[EBC 93] EBCIOGLU K., « An Expert System for Harmonizing Four-Part Chorales », dans S.M. Schwanauer et D.A. Levitt (dir.), *Machine Models of Music*, MIT Press, Cambridge, p. 385-401, 1993.

[FUX 65] FUX J.J., *The Study of Counterpoint from Johann Joseph Fux's Gradus ad Parnassum*, W.W. Norton & Company, New York,1965.

[GUE 98] GUÉNÉGO J.-L, Une application de résolution de contraintes en harmonie tonale, IRCAM/Université Paris VI, http://www.ircam.fr/equipes/repmus/Rapports/JLGuenego98, 1998.

[HEN 96] Henz M., *et al*., « CompoZe- Intention-Based Music Composition Through Constraint Programming », *8th IEEE International Conference on Tools with AI*, Toulouse, France, 1996.

[HIL 92] HILD H., FEULNER J., MENZEL W., « HARMONET: A Neural Net for Harmonizing Chorals in the Style of J.S. Bach », dans R.P. Lippmann, J.E. Moody, D.S. Touretzky (dir.), *Advances in Neural Information Processing 4 (NIPS 4)*, p. 267-274, Morgan Kaufmann, 1992.

[HILL 93] HILLER L., ISAACSON L., « Musical Composition with a High-Speed Digital Computer », dans M. Schwanauer et D. Levitt (dir.), *Machines Models of Music*, MIT Press, Cambridge, p. 9-21, 1993.

[JAF 87] JAFFAR J., LASSEZ J.L., « Constraint Logic Programming », *IEEE 4th International Conference on Logic Programming*, Melbourne, 25-29 mai 1987.

[LEV 93] LEVITT D.A., « A Representation of Musical Dialects », dans S.M. Schwanauer et D.A. Levitt (dir.), *Machine Models of Music*, MIT Press, Cambridge, p. 456-469, 1993.

[LEV 81] LEVITT D., « A Melody Description System for Jazz Improvisation », Masters Thesis, MIT Artificial Intelligence Lab, Cambridge, 1981.

[MAC 77] MACKWORTH A.K., « Consistency in networks of relations », *Artificial Intelligence*, vol. 8 (1), p. 99-118, 1977.

[OVA 92a] OVANS R., Efficient Music Composition via Consistency techniques, Rapport technique CSS-IS TR 92-02, Université Simon Fraser, Canada, 1992.

[OVA 92b] OVANS R., DAVISON R., An Interactive Constraint-Based Expert Assistant for Music Composition. Ninth Canadian Conference on Artificial Intelligence, Université British Columbia, Vancouver, p. 76-81, 1992.

[PAC 95a] PACHET F., ROY P., « Integrating constraint satisfaction techniques with complex object structures », *15th Annual Conference of the British Computer Society Specialist Group on Expert Systems, ES'95*, Cambridge, p. 11-22, 1995.

[PAC 95b] PACHET F., ROY P., « Mixing constraints and objects: a case study in automatic harmonization », *TOOLS Europe'95*, Prentice-Hall, p. 119-126, 1995.

[PAC 00] PACHET F., ROY P., CAZALY D., « A Combinatorial Approach to Content-Based Music Selection », *IEEE Multimedia*, 7(1):44-51, mars 2000.

[PHO 98] PHON-AMNUAISUK S., WIGGINS G., « The Four-Part harmonization problem: A comparison between genetic algorithms and a Rule-Based system », *Proceedings of the AISB'99 Symposium on Musical Creativity, AISB Symposium*, Edinbourg, 1998.

[PON 99] PONSFORD D., WIGGINS G., MELLISH C., « Statistical Learning of Harmonic Movement », *Journal of New Music Research*, 28:2, p. 150-177, 1999.

[RAM 85] RAMEAU J.-P., *Treatise on Harmony*, Dover Publications, New York, 1985.

[RAMI 98] RAMIREZ R., PERALTA J., « A constraint-based melody harmonizer », *ECAI'98 Workshop on Constraints and Artistic Applications*, Brighton, 1998.

[ROT 68] ROTHGEB J., Harmonizing the Unfigured Bass: a Computational Study, Thèse de doctorat, Université d'Indiana, 1968.

[ROY 98] ROY P., Satisfaction de contraintes et programmation par objets, Thèse de doctorat, Université Paris 6, 1998.

[RUE 97] RUEDA C., VALENCIA F., « Improving forward-checking with delayed evaluation », *Proceedings of CLEI'97*, Santiago, Chili, 1997.

[RUE 98] RUEDA C., LINDBERG M., LAURSON M., BLOCH G., ASSAYAG G., « Integrating Constraint Programming in Visual Musical Composition Languages », *ECAI Workshop* on Constraints for Artistic Applications, Brighton, 1998.

[SCH 93] SCHOENBERG A., Theory of Harmony, Université de California Press, 1993.

[SCHO 89] SCHOTTSTAEDT B., « Automatic species counterpoint. Tech. Rep. STAN-M-19, Stanford University CCRMA. A short report with source code appeared », dans M.V. Mathews et J.R. Pierce (dir.), *Current Directions in Computer Music Research*, MIT Press, Cambridge, 1989.

[STE 79] STEELS L., *Reasoning modeled as a Society of Communicating Experts*, AI-TR-542. Artificial Intelligence Lab, MIT, Cambridge, 1979.

[STE 86] STEELS L., *Learning the craft of musical composition*, ICMC, The Hague, 1986

[TRU 03] TRUCHET C., ASSAYAG G., CODOGNET P., « OMClouds, a Heuristic Solver for Musical Constraints », *MIC03, Metaheuristics International Conference,* Kyoto, Japon, août 2003.

[TSA 91] TSANG C.P., AITKEN M., *Harmonizing music as a discipline of constraint logic programming*, ICMC, Montréal, 1991.

[VOS 78] VOSS R., CLARKE J., « 1/f noise in Music », *Journal of the Acoustical Society of America*, 63, p. 258-263, 1978.

[WIG 98] WIGGINS G., « The Use of Constraint Systems for Musical Composition », *ECAI'98 Workshop on Constraints and Artistic Applications*, Brighton, 1998.

[ZIL 01] ZILS A., PACHET F., « Musical Mosaicing », *Proceedings of DAFX 01*, Université de Limerick, 2001.

[ZIM 01] ZIMERMANN D., « Modelling Musical Structures », *Constraints Journal*, Kluwer Publisher, 6(1):7-19, 2001.

Chapitre 11

L'analyse de Fourier

La manière la plus simple de représenter un signal audio est certainement de considérer son amplitude en fonction du temps $s(t)$, où t est le temps exprimé en secondes. C'est la classique représentation temporelle.

Cependant, une autre (et extrêmement utile, car plus proche de la perception) manière de représenter un signal audio est la représentation fréquentielle (ou spectrale), qui fait intervenir des fréquences et des amplitudes en fonction du temps (se reporter par exemple à [ORF 96] pour une introduction détaillée).

La synthèse et la manipulation de sons numériques nécessitent de bons modèles sonores. Les modèles spectraux fournissent des représentations générales dans lesquelles de telles opérations peuvent être effectuées d'une façon très naturelle et expressive sur le plan musical. Afin d'imiter fidèlement ou de transformer des sons existants, ces modèles nécessitent une méthode d'analyse pour extraire les paramètres spectraux de sons classiquement enregistrés dans le modèle temporel, c'est-à-dire l'amplitude du signal audio fonction du temps. La précision de la méthode d'analyse est extrêmement importante puisque la qualité perçue pour les sons spectraux résultants dépend principalement d'elle. L'intérêt principal d'une méthode d'analyse précise, fournissant des paramètres exacts aux modèles spectraux, est de permettre des transformations musicales sur le son toujours plus profondes tout en minimisant les déformations du son dues aux artefacts provenant de l'analyse.

Chapitre rédigé par Sylvain MARCHAND.

Concevoir une méthode d'analyse précise n'est pas chose facile. Ce chapitre présente quelques-unes des méthodes d'analyse proposées pour extraire les paramètres spectraux de sons à l'origine exprimés dans le modèle temporel. Pour que ces méthodes soient efficaces, il faut toutefois restreindre le modèle spectral considéré.

Ce chapitre se limitera aux modèles dits sinusoïdaux. Les sons sont alors constitués de partiels, oscillations sinusoïdales de fréquences et d'amplitudes à déterminer. De nombreux modèles sinusoïdaux ont été proposés ces dernières années et ont fait la preuve de leur utilité pratique dans des architectures logicielles comme Lemur [FIT 96], SMS [SERR 97] ou *InSpect* [MAR 99, MAR 00], grâce à la puissance toujours croissante des ordinateurs modernes. Toutefois, les modèles sinusoïdaux sont dérivés des travaux de Joseph Fourier au XIX^{e} siècle.

Certaines méthodes d'analyse supposent que le son est harmonique et donc que les partiels sont espacés régulièrement en fréquence. Mais ce serait se priver des sons polyphoniques et nous ne ferons pas cette supposition ici.

En revanche, les sons considérés dans ce chapitre doivent nécessairement avoir un faible niveau de bruit, ce qui est le cas pour de nombreux sons naturels clairs, après leur phase d'attaque toutefois.

Une autre restriction est que les partiels doivent être suffisamment espacés en fréquence. Pour un son s donné, il doit y avoir une distance minimale $d > 0$ telle que :

$$\min_{i \neq j,\, t} \{|f_j(t) - f_i(t)|\} > d$$

où les f_p sont les fréquences des partiels. Cette condition, qui interdit également le « croisement » de deux partiels, est une hypothèse raisonnable qui se vérifie pour un grand nombre de sons. La nécessité de cette condition sera mise en évidence en section 11.3.

La section 11.1 présente la transformée de Fourier classique. La section 11.2 met alors en évidence ses principales limitations et imprécisions tandis que la section 11.3 décrit quelques méthodes intéressantes qui permettent d'améliorer grandement la précision de l'analyse de Fourier classique.

Afin de bien comprendre ce chapitre, il est nécessaire d'avoir des connaissances de base en mathématiques [MOO 78a, MOO 78b, MOO 85] et traitement du signal [MOOR 85, OPP 89, ORF 96] appliqués à l'informatique musicale. Les notions élémentaires de traitement du signal et d'algorithmique de niveau deuxième cycle universitaire doivent amplement suffire. Les ouvrages cités précédemment pourront être consultés en complément si nécessaire.

11.1. La transformée de Fourier

11.1.1. *Notations*

Dans la suite de ce chapitre, la plupart des expressions mathématiques font appel aux nombres complexes. On y utilise indifféremment les notations cartésienne ou polaire.

Soit le nombre complexe $c = x + i\,y = ae^{i\,\phi}$, exprimé dans ces deux représentations. Les nombres x, y, a et ϕ sont des réels, e désigne la fonction exponentielle complexe (notation d'Euler, voir plus loin) et i est l'imaginaire pur (complexe tel que $i^2 = -1$).

Dans la notation cartésienne, x (respectivement y) est la partie réelle (respectivement imaginaire) du nombre complexe c, notée $\mathrm{Re}(c)$ (respectivement $\mathrm{Im}(c)$).

Dans la notation polaire, a et ϕ sont, respectivement, le module (l'amplitude) et l'argument (la phase) du nombre complexe c.

Il est important de pouvoir passer facilement d'une représentation à l'autre. On a $x = a\cos(\phi)$ et $y = a\sin(\phi)$ (par la formule d'Euler) et, réciproquement, $a = \sqrt{(x^2 + y^2)}$ et $\phi = \arctan(\frac{y}{x}) + \alpha(x) \cdot \pi$, où $\alpha(x)$ vaut 1 si x est négatif et 0 sinon.

11.1.2. *Définitions*

La transformée de Fourier et son inverse sont des transformées mathématiques qui permettent, respectivement, de passer du domaine temporel au domaine fréquentiel (spectral) et inversement. Il existe en fait plusieurs transformées de Fourier, selon que l'on se place dans le cas où le temps est continu ou discret, si l'on s'intéresse aux pulsations plutôt qu'aux fréquences, ou encore si le signal considéré est de support fini ou non.

Si s (respectivement S) est l'expression du signal dans le domaine temporel (respectivement fréquentiel), alors la transformée de Fourier continue et son inverse sont donnés par les formules suivantes :

$$S(\Omega) = \int_{-\infty}^{+\infty} s(t)\, e^{-i\Omega t}\, dt \qquad [11.1]$$

$$s(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} S(\Omega)\, e^{+i\Omega t}\, d\Omega \qquad [11.2]$$

La pulsation Ω (exprimée en radians par seconde) est une fonction de la fréquence f (exprimée en hertz, Hz) :

$$\Omega = 2\pi f \quad [11.3]$$

Par conséquent, la transformée de Fourier et son inverse peuvent également être définis en utilisant la fréquence :

$$S(f) = \int_{-\infty}^{+\infty} s(t)\, e^{-i2\pi ft}\, dt \quad [11.4]$$

$$s(t) = \int_{-\infty}^{+\infty} S(f)\, e^{+i2\pi ft}\, df \quad [11.5]$$

Lorsque l'on considère que le temps est discret, on note :

$$s[k] = s(kT_s) \quad [11.6]$$

où la période d'échantillonnage T_s (exprimée en secondes) est l'inverse de la fréquence d'échantillonnage F_s. Pour un signal s en temps discret, les expressions de la transformée de Fourier en temps discret et de son inverse sont respectivement :

$$S(\omega) = \sum_{n=-\infty}^{+\infty} s[n]\, e^{-i\omega n} \quad [11.7]$$

$$s[k] = \frac{1}{2\pi} \int_{-\pi}^{+\pi} S(\omega)\, e^{+i\omega k}\, d\omega \quad [11.8]$$

$S(\omega)$ est une fonction 2π-périodique et ω est une fonction de la fréquence :

$$\omega = \frac{2\pi f}{F_s} \quad [11.9]$$

La transformée de Fourier en temps discret et son inverse peuvent alors être définis en utilisant la fréquence :

$$S(f) = \sum_{n=-\infty}^{+\infty} s[n]\, e^{-i2\pi \frac{fn}{F_s}} \quad [11.10]$$

$$s[k] = \frac{1}{F_s} \int_{-F_s/2}^{+F_s/2} S(f)\, e^{+i2\pi \frac{fk}{F_s}}\, df \quad [11.11]$$

Cette fois, $S(f)$ est une fonction F_s-périodique.

Si, de plus, le signal s est de longueur N finie, alors la transformée de Fourier discrète est un cas particulier de la transformée de Fourier en temps discret. Soit $\mathcal{B} = \frac{F_s}{N}$ et :

$$f_k = k\mathcal{B} \quad [11.12]$$

$$\omega_k = \frac{2\pi f_k}{F_s} = \frac{2\pi k}{N} \quad [11.13]$$

Les valeurs f_k et ω_k pour $k \in [0, N[$ sont les fréquences de la transformée de Fourier discrète exprimées respectivement en Hz ou en cycles par échantillon. Si on note $S[k] = S(\omega_k)$ (pour les équations [11.7] et [11.8]) ou $S[k] = S(f_k)$ (pour les équations [11.10] et [11.11]), la transformée de Fourier discrète et son inverse peuvent être respectivement définis par les équations suivantes :

$$S[m] = \sum_{n=0}^{N-1} s[n] e^{-i2\pi \frac{mn}{N}} \quad [11.14]$$

$$s[k] = \frac{1}{N} \sum_{n=0}^{N-1} S[n] e^{+i2\pi \frac{kn}{N}} \quad [11.15]$$

Un facteur de normalisation $\frac{2}{N}$ est souvent utilisé dans l'équation [11.14] pour obtenir la valeur exacte de l'amplitude des sinusoïdes directement dans le spectre (voir plus loin). Dans ce cas, $\frac{1}{2}$ remplace $\frac{1}{N}$ dans l'équation [11.15].

La transformée de Fourier discrète calcule à chaque instant les valeurs instantanées de l'amplitude et de la phase du signal pour chacune des N composantes fréquentielles (les casiers, *bins* en anglais), espacées de $\mathcal{B} = \frac{F_s}{N}$ Hz.

Il existe des algorithmes de transformée de Fourier rapide (*Fast Fourier Transform*, FFT) [COO 65] qui calculent la transformée de Fourier discrète avec une complexité de seulement $O(N \log(N))$, au lieu de $O(N^2)$ pour les équations [11.14] et [11.15]. Notons toutefois que, pour être efficaces, ces algorithmes exigent, la plupart du temps, que N soit une puissance de 2. Pour les réalisations pratiques, la bibliothèque FFTW [FRI 03], développée au MIT, est très performante.

11.1.3. *Spectres*

Le spectre S d'un signal s est une fonction complexe de la fréquence. Le spectre d'amplitude (respectivement de phase) du spectre complexe S consiste en l'amplitude (respectivement la phase) des valeurs complexes composant ce spectre.

Le signal s est réel si et seulement si son spectre S est conjugué-symétrique ($\forall x$, $S(-x) = \bar{S}(x)$: la partie réelle de S est une fonction paire, sa partie imaginaire est une fonction impaire).

Une simple impulsion de Dirac dans le domaine temporel donne lieu à un spectre uniformément plat dans le domaine fréquentiel, tandis qu'une sinusoïde (fonction dont le support est infini dans le domaine temporel) engendre une impulsion de Dirac (un « pic ») à la fréquence correspondante dans le domaine fréquentiel.

11.1.3.1. *Opérations sur les spectres*

Les domaines temporel et fréquentiel sont intimement liés. L'addition ($+$) dans le domaine temporel et l'addition dans le domaine fréquentiel sont les mêmes opérations. De plus, un signal s est nul ($\forall t$, $s(t) = 0$) si et seulement si son spectre S est nul ($\forall f$, $S(f) = 0$).

Les choses se compliquent un peu avec la multiplication ($\times$), nécessitant la définition d'une nouvelle opération : la convolution.

La convolution $*$ est définie, suivant que le temps est continu ou discret, par les expressions suivantes :

$$x(t) * y(t) = \int_{-\infty}^{+\infty} x(u)\, y(u-t)\, du \qquad [11.16]$$

$$x[n] * y[n] = \sum_{k=-\infty}^{+\infty} x[k]\, y[n-k] \qquad [11.17]$$

Le théorème de convolution dit que la multiplication (respectivement la convolution) dans le domaine temporel est équivalente à la convolution (respectivement la multiplication) dans le domaine fréquentiel, ce qui se résume ainsi, avec la notation classique en majuscules pour les spectres et en minuscules pour les signaux :

$$x \times y \leftrightarrow X * Y \qquad [11.18]$$

$$x * y \leftrightarrow X \times Y \qquad [11.19]$$

11.1.3.2. *Technique du* zero-padding

La technique du *zero-padding* (littéralement « bourrage de zéros ») consiste, dans le domaine temporel, à augmenter la taille du signal s en ajoutant des 0 (zéros) au début (ou à la fin) du signal.

Dans le domaine fréquentiel, le *zero-padding* consiste à augmenter artificiellement la taille du spectre, en ajoutant des zéros pour des fréquences qui ne se trouvaient pas dans le spectre d'origine.

Le *zero-padding* dans le domaine temporel (respectivement fréquentiel) est équivalent à l'interpolation (suréchantillonnage) dans le domaine fréquentiel (respectivement temporel). L'interpolation dans le domaine fréquentiel ne fournit pas un spectre plus détaillé, mais une version plus lissée de ce dernier.

11.1.4. *Modèles sinusoïdaux*

Le théorème de Fourier assure que toute fonction périodique peut être exprimée sous la forme d'une somme de sinusoïdes d'amplitudes diverses et de fréquences liées harmoniquement. Pour les sons quasi périodiques, ces amplitudes et fréquences varient lentement dans le temps. Les *partiels* correspondent à ces oscillations quasi sinusoïdales, contrôlées dans le temps en fréquence et en amplitude. Dans les modèles sinusoïdaux, le signal audio s dans le domaine temporel est donné par les équations suivantes :

$$s(t) = \sum_{p=1}^{P} \mathrm{osc}\big(f_p(t), a_p(t), \phi_p(0)\big) \qquad [11.20]$$

où P est le nombre (fini) de partiels et :

$$\mathrm{osc}\big(f_p(t), a_p(t), \phi_p(0)\big) = a_p(t)\cos\big(\phi_p(t)\big) \qquad [11.21]$$

avec :

$$\frac{d\phi_p}{dt}(t) = 2\pi f_p(t) \quad \text{c'est-à-dire} \quad \phi_p(t) = \phi_p(0) + 2\pi \int_0^t f_p(u)\,du \qquad [11.22]$$

Les fonctions f_p, a_p et ϕ_p sont, respectivement, la fréquence, l'amplitude et la phase instantanées du p-ième partiel.

Les modèles sinusoïdaux sont issus de la synthèse additive (voir [MOOR 77]), la toute première technique de modélisation spectrale. Ces modèles sont aujourd'hui très répandus, que ce soit la représentation sinusoïdale de McAulay et Quatieri [MCA 86] présentée ici ou son extension pour la prise en considération des bruits dans SMS [SERR 89, SERR 90, SERR 97] et aussi des transitoires dans STN [VER 98a, VER 98b]. De nombreux logiciels implantant ces modèles sont disponibles. Citons, entre autres, AudioSculpt [IRC 96] de l'IRCAM, Lemur [FIT 96], SMS ou encore InSpect [MAR 99, MAR 00] du SCRIME.

A première vue, il peut paraître curieux de parler de modèles sinusoïdaux quand les équations associées ne font intervenir que des fonctions cosinus. En fait, dans l'équation [11.21], la fonction cosinus peut très bien être remplacée par la fonction sinus, puisque ces fonctions sont les mêmes à un déphasage de $\pi/2$ près :

$$\cos(x) = \sin\Big(x + \frac{\pi}{2}\Big) \qquad [11.23]$$

$$\sin(x) = \cos\Big(x - \frac{\pi}{2}\Big) \qquad [11.24]$$

En revanche, l'hypothèse que les fréquences et les amplitudes des partiels varient lentement est très importante. Considérons par exemple un son s quelconque. La solution $P = 1$, $f_1(t) = 0$, $a_1(t) = s(t)$ vérifie de manière triviale les équations du

modèle, mais comme a_1 varie très rapidement dans le temps, cette solution est rejetée car les hypothèses de stationnarité du modèle ne sont pas respectées. Toutefois, cette notion de « varier lentement dans le temps » doit être clarifiée.

Plus formellement, les fonctions a_p et f_p sont elles-mêmes des signaux dans le domaine temporel. Les spectres de ces signaux sont limités en fréquence par une certaine valeur bien inférieure à la plus petite des fréquences f_p. Plus précisément, ils ne doivent pas contenir de fréquences supérieures à une fréquence $F_{\max}$ située aux alentours de 20 Hz. En effet, les variations des paramètres du modèle doivent rester inaudibles, sous peine de voir surgir des phénomènes non linéaires de modulation, indésirables.

Les partiels se traduisent par des « pics » (des maximums locaux) dans le spectre d'amplitude (voir figure 11.1) et correspondent à des oscillations sinusoïdales dans le domaine temporel.

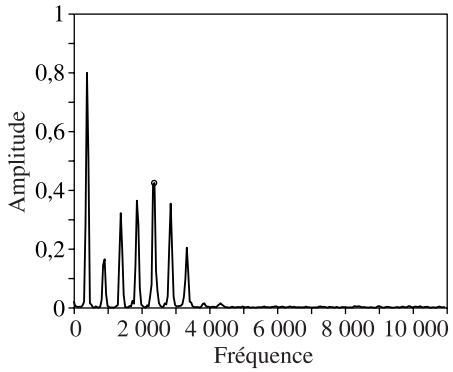


Figure 11.1. *Spectre d'amplitude résultant d'une transformée de Fourier rapide (FFT)*

11.1.5. *Transformée de Fourier à court terme*

La transformée de Fourier convertit le signal temporel (amplitude en fonction du temps) en une représentation fréquentielle (amplitude en fonction de la fréquence). Elle donne l'image spectrale de la totalité du son et le signal se retrouve tout entier à l'intérieur d'un unique spectre.

Ce spectre est en accord avec la perception uniquement dans le rare cas des sons stationnaires. La plupart des sons évoluant dans le temps, les modèles sonores doivent avoir des paramètres fonctions du temps et, par conséquent, les méthodes d'analyse doivent être capables de produire des résultats qui dépendent également du temps.

Il suffit souvent de recycler des algorithmes indépendants du temps en se contentant de les répéter sur de petites portions du signal prises à des instants successifs. Ce type de traitement est appelé une analyse à court terme.

Parmi les analyses à court terme les plus utilisées se trouve la transformée de Fourier à court terme [ALL 77, POR 80], qui est au centre du vocodeur de phase [DOL 86, MOOR 78, SER 97].

Cette transformée constitue une version dépendante du temps de la transformée de Fourier vue précédemment. Elle produit une suite de spectres dits « à court terme » pris pour des segments temporels x successifs, souvent avec un facteur de recouvrement, qui sont des petits morceaux du signal s dans le domaine temporel. En pratique, le signal s à analyser est discret et résulte d'un échantillonnage uniforme à la fréquence d'échantillonnage F_s.

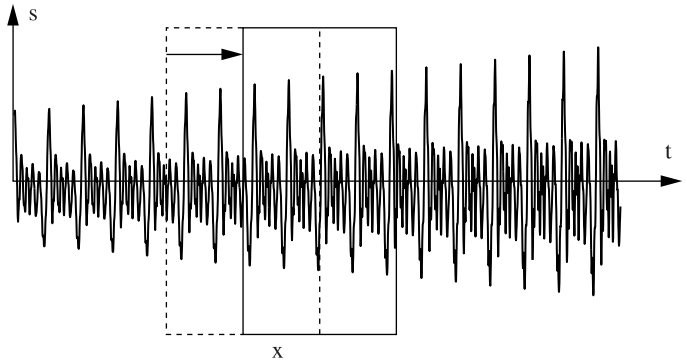


Figure 11.2. *Une fenêtre d'analyse glisse d'un bout à l'autre du signal s. A chaque arrêt, un segment x de N échantillons consécutifs est extrait. Le facteur de recouvrement est ici de $1/2$: la moitié des échantillons sont partagés entre deux segments successifs*

Une fenêtre d'analyse glisse en parcourant la totalité du signal, comme illustré sur la figure 11.2. Habituellement, la largeur N de la fenêtre d'analyse ainsi que le facteur de recouvrement utilisés sont constants. Alors, pour chaque segment x de N échantillons consécutifs, une fenêtre d'analyse (discrète) w est appliquée (multipliée) à x et la transformée de Fourier (discrète) X est calculée en utilisant l'équation [11.25] :

$$X[m] = \frac{2}{N} \sum_{n=0}^{N-1} w[n]\, x[n]\, e^{-i\frac{2\pi}{N}nm} \qquad [11.25]$$

Dans l'équation [11.25], une fenêtre d'analyse w est multipliée au segment x, puis le résultat est lui-même multiplié par des fonctions exponentielles complexes.

Puisque $e^{i\phi} = \cos(\phi) + i\sin(\phi)$, les parties réelles et imaginaires de ces exponentielles complexes sont respectivement des fonctions cosinus et sinus. Une autre façon d'interpréter l'équation [11.25] est de considérer que ces fonctions cosinus et sinus sont multipliées par la fenêtre d'analyse w et que les produits obtenus sont alors multipliés avec le signal x. Les produits de la fenêtre d'analyse avec ces oscillations cosinusoïdales ou sinusoïdales ressemblent à des ondelettes. Il s'agit en fait de grains de Gabor (si toutefois la fenêtre w est une gaussienne), comme celui illustré en figure 11.3. Pour cette raison, cette transformée est aussi appelée transformée de Gabor quand w est une gaussienne [ARF 91].

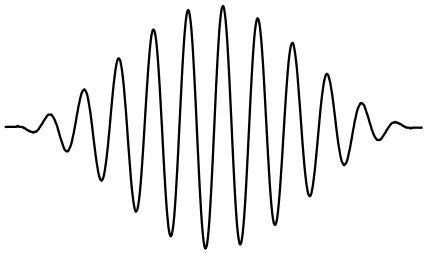


Figure 11.3. *Exemple de grain de Gabor*

L'utilisation de la fenêtre d'analyse w est obligatoire. Ne pas utiliser de fenêtre reviendrait en fait à utiliser la fenêtre rectangulaire (voir section 11.2), qui vaut 1 sur l'intervalle $[0; N[$ et 0 en dehors. C'est souvent un très mauvais choix. De meilleures fenêtres d'analyse sont présentées au paragraphe 11.2.4.

Le résultat est un spectre (à court terme), comme celui représenté en figure 11.1. En plus de la fenêtre d'analyse w, le facteur de normalisation $\frac{2}{N}$ a été ajouté à l'équation [11.14], comme indiqué précédemment au paragraphe 11.1.2.

L'analyse de McAulay et Quatieri [MCA 86] suit alors les maximums locaux tout au long de la succession des spectres à court terme afin de reconstruire les évolutions des partiels.

11.2. Imprécisions de la transformée de Fourier classique

La transformée de Fourier est souvent utilisée pour extraire la partie déterministe (non bruitée) des sons, modélisée sous la forme d'un ensemble de partiels dont les paramètres (fréquences f_p et amplitudes a_p) varient lentement dans le temps.

Dans la suite de ce chapitre, on se place dans le cas des signaux quasi stationnaires. Ce sont des signaux non stationnaires, les paramètres des partiels qui les composent

évoluant lentement dans le temps, mais ces évolutions sont suffisamment lentes pour que l'on puisse considérer ces paramètres comme constants sur un petit intervalle de temps correspondant à N échantillons.

Une variation des paramètres fréquence et amplitude des partiels au cours de la fenêtre d'analyse engendre une distortion de la forme du spectre W de la fenêtre d'analyse w (lire [MAR 86, MAS 95, MAS 98] pour de plus amples détails). La phase de W est principalement affectée (voir figure 11.4). Toutefois, la fréquence et l'amplitude du partiel mesuré restent conformes à leurs valeurs instantanées au centre de la fenêtre d'analyse, à condition toutefois que les variations des paramètres restent raisonnables.

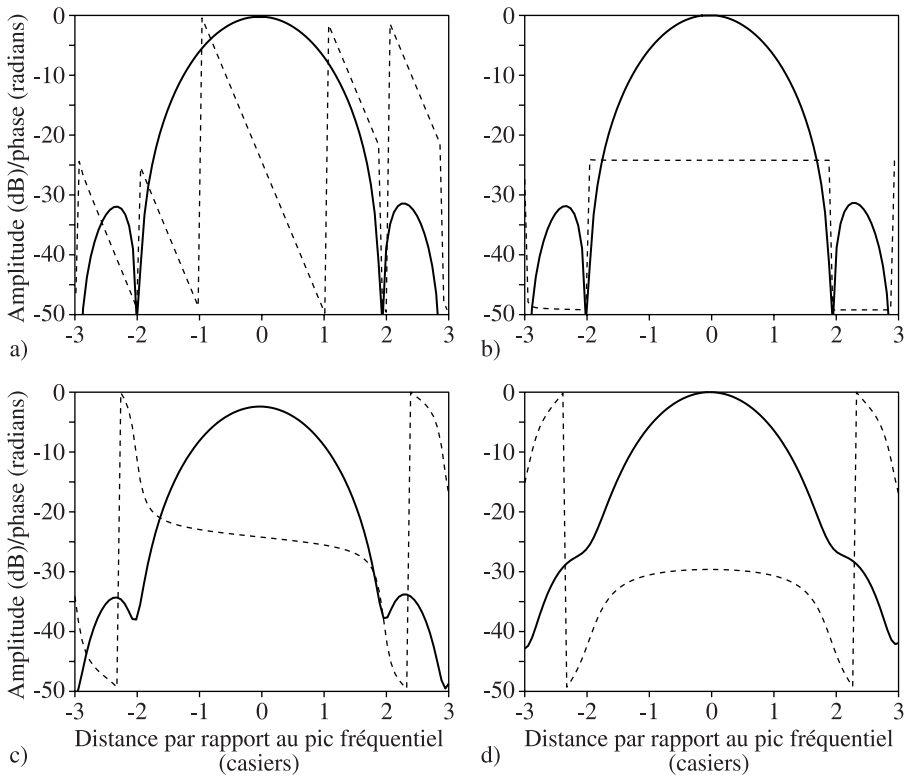


Figure 11.4. *Forme du spectre de la fenêtre de Hann dans différents cas de figure. Le spectre d'amplitude est représenté en trait plein tandis que le spectre de phase (compris entre $-\pi$ et π) est tracé en pointillés. En haut se trouve le cas stationnaire (fréquence et amplitude du partiel constantes), dans le cas du fenêtrage classique (phase linéaire, a) et dans celui du fenêtrage dit* zero-phase *(b). En bas se trouve le cas non stationnaire : c) une variation d'amplitude de $+6$ dB pendant la durée de la fenêtre d'analyse, d) une variation de fréquence de 1 casier ($\mathcal{B}$ Hz).*

L'approche classique pour estimer les fréquences et les amplitudes des partiels est d'extraire les maximums locaux (les « pics ») à l'intérieur du spectre d'amplitude et de déterminer la fréquence, l'amplitude et éventuellement la phase en chacun de ces maximums.

Soit m l'indice d'un de ces maximums à l'intérieur du spectre à court terme X ; on a donc $|X[m-1]| < |X[m]| < |X[m+1]|$. Il peut sembler logique, *a priori*, de retenir pour le partiel correspondant des fréquence, amplitude et phase instantanées valant respectivement $f = m\mathcal{B}$, $a = |X[m]|$ et $\phi = \arg(X[m])$.

Ce serait malheureusement commettre une grave erreur, tant les imprécisions sur ces paramètres seront grandes dans ce cas. Il faut tout d'abord penser à normaliser l'amplitude a, en divisant $|X[m]|$ par $\sum_n w[n]$, qui est la contribution de la fenêtre w à l'énergie (l'amplitude pour être précis) du partiel. Mais cela ne suffit pas, loin de là. La figure 11.5 fournit une illustration claire et synthétique de deux phénomènes. Elle montre le résultat de la transformée de Fourier à court terme sur une unique oscillation sinusoïdale dont la fréquence augmente linéairement dans le temps tandis que son amplitude reste constante.

Les équations de la fréquence et de l'amplitude en fonction du temps de l'unique partiel de cet exemple synthétique sont :

$$f_1(t) = F_0 + \frac{F_1 - F_0}{D}t \qquad [11.26]$$

$$a_1(t) = A \qquad [11.27]$$

où D est la durée du son s exprimée en secondes et $A > 0$, $F_0 > 0$ et $F_1 > F_0$ sont des constantes.

En prenant arbitrairement $\frac{\pi}{2}$ (pour annuler la fonction cosinus) pour valeur initiale de la phase et en considérant les équations [11.20], [11.21] et [11.22] des modèles sinusoïdaux, on calcule :

$$\phi_1(t) = \frac{\pi}{2} + 2\pi \int_0^t f_1(u)\,du = \frac{\pi}{2} + 2\pi\Big(F_0 + \frac{F_1 - F_0}{2D}t\Big)t \qquad [11.28]$$

et finalement l'expression du signal s :

$$s(t) = A\cos\Big(2\pi\Big(F_0 + \frac{F_1 - F_0}{2D}t\Big)t\Big) \qquad [11.29]$$

En utilisant l'analyse de Fourier à court terme classique sur le son s, on retrouve bien un unique partiel. Les évolutions de sa fréquence et de son amplitude en fonction du temps sont illustrées sur la figure 11.5.

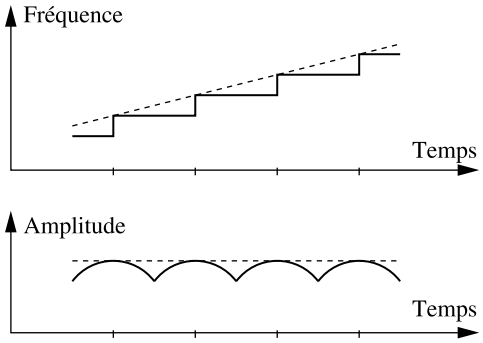


Figure 11.5. *Les évolutions en fréquence (en haut) et en amplitude (en bas), en fonction du temps, pour une unique oscillation sinusoïdale dont la fréquence augmente linéairement tandis que son amplitude reste constante. Les évolutions théoriques du signal d'origine s (voir équations [11.26] et [11.27]) sont tracées en pointillés, tandis que les valeurs trouvées par l'analyse de Fourier à court terme classique sont représentées en trait plein. Les marques sur l'axe du temps indiquent quand la fréquence de l'oscillation change de casier de la transformée de Fourier discrète.*

Toutefois, la courbe représentant la fréquence mesurée n'est pas une ligne droite comme elle devrait l'être (d'après l'équation [11.26]), mais correspond à une fonction en escalier, due à la discrétisation du spectre (d'où une erreur en fréquence).

De plus, la courbe représentant l'amplitude mesurée n'est pas plate, et donc ne correspond pas à une fonction constante (voir équation [11.27]), mais présente une succession de bosses, dues à la forme du lobe principal du spectre de la fenêtre d'analyse (d'où une erreur en amplitude).

Ceci explique pourquoi le vocodeur de phase classique se comporte souvent si mal en présence de sons présentant un vibrato (variation sinusoïdale de la fréquence fondamentale) ou de trémolo (variation sinusoïdale de l'amplitude).

De plus, si l'on n'y prend pas garde, la phase instantanée mesurée pour les partiels est également fausse.

11.2.1. *Imprécision en fréquence*

L'imprécision en fréquence vient du fait que la transformée de Fourier discrétise le spectre de 0 Hz (composante continue) à $\frac{F_s}{2}$ Hz (fréquence de Nyquist) par pas de $\mathcal{B} = \frac{F_s}{N}$ Hz, où F_s est la fréquence d'échantillonnage utilisée pour le signal s et N est la taille de la transformée de Fourier. Pour réduire cette erreur, on pourrait essayer de réduire F_s ou d'augmenter N.

Malheureusement, réduire F_s appauvrirait le contenu fréquentiel du signal s, puisque le théorème d'échantillonnage de Shannon-Nyquist dit que l'on ne peut pas espérer stocker dans s des fréquences supérieures à $\frac{F_s}{2}$.

Si une valeur trop petite pour N implique une imprécision trop grande pour la fréquence, une valeur trop grande pour N implique une fenêtre d'observation très longue (puisque N est la taille de la fenêtre d'analyse en échantillons, cette fenêtre dure donc $\frac{N}{F_s}$ secondes) et il est alors très probable que les paramètres du partiel auront le temps de varier pendant la période d'observation. Au final, on obtient des paramètres instantanés « flous », comme ce que l'on pourrait obtenir en prenant en photo un sujet mobile avec un temps d'exposition trop grand. Une bonne résolution temporelle implique une mauvaise précision en fréquence et *vice versa*. Une valeur de N trop grande produit une sensation de réverbération, alors qu'une valeur trop petite donne lieu à des sons qualifiés de « métalliques ». C'est le célèbre compromis temps-fréquence de la transformée de Fourier à court terme. Choisir « la bonne valeur » de N est alors chose ardue – et cette valeur peut très bien ne pas exister pour des sons complexes.

11.2.2. *Imprécision en amplitude*

Dans l'équation [11.25], le spectre à court terme obtenu est en fait $W * X$, d'après le théorème de convolution vu au paragraphe 11.1.3 et puisque l'on s'intéresse en fait au signal $w \times x$. Extraire un segment x du signal s peut être vu comme la multiplication de s par la fenêtre rectangulaire qui vaut 1 sur l'intervalle de temps considéré et 0 en dehors. Négliger la fenêtre et enlever w de l'équation [11.25] reviendrait en fait à utiliser cette fenêtre rectangulaire, ce qui est un très mauvais choix (voir plus loin).

Si x est une oscillation purement sinusoïdale d'amplitude a et de fréquence f, X est une raie spectrale : $|X(f)| = a$ et $|X(f')| = 0$ pour $f' \neq f$. Mais puisque le spectre de la fenêtre W est de la forme de ceux de la figure 11.8, la convolution $W * X$ est loin d'être une raie spectrale comme on le souhaiterait. La transformée de Fourier (discrète) à court terme d'un signal discret de taille N, avec la fenêtre d'analyse w, d'une sinusoïde parfaitement stationnaire est le spectre W de la fonction de fenêtrage w, centré sur la fréquence de la sinusoïde, et échantillonné aux fréquences correspondant aux casiers de la transformée de Fourier discrète. Ce spectre est également multiplié par l'amplitude de la sinusoïde et déphasé par la phase instantanée de la sinusoïde au centre de la fenêtre temporelle considérée.

La fenêtre w a un grand impact sur la précision de l'analyse aussi bien en temps qu'en fréquence. Ses effets sur le spectre X doivent être réduits autant que faire se peut.

En pratique, dans la plupart des cas, W comporte des « lobes » et il est souhaitable que les lobes secondaires soient négligeables par rapport au lobe principal.

Malheureusement, ce rapport entre les lobes secondaires et le lobe principal est inversement proportionnel à la largeur du lobe principal, qui détermine la résolution en fréquence de la fenêtre d'analyse. Choisir la « bonne » fenêtre est alors chose ardue. Une discussion sur le choix de la fenêtre est abordée au paragraphe 11.2.4.

Une bonne résolution fréquentielle requiert un lobe principal étroit, mais alors la courbure de la forme caractéristique du lobe est plus forte et il en résulte une distorsion dans le spectre d'amplitude (voir figure 11.5). C'est la raison pour laquelle l'amplitude mesurée dans le cas du signal s de l'équation [11.29] est si déformée quand sa fréquence augmente, le partiel passant de casier en casier (voir figure 11.5).

Toute méthode d'analyse qui se respecte devrait tenir compte de ce phénomène. Naturellement, de si petites déformations ne sont en général pas audibles, mais elles peuvent le devenir dramatiquement après des transformations musicales sur le son. Le tableau 11.1 montre que les imprécisions en amplitude pour les fenêtres d'analyse les plus communément utilisées sont assez importantes. Pour minimiser cette imprécision en amplitude, un lobe principal « plat » est nécessaire. Mais ce n'est possible qu'avec un lobe principal très large – et donc une résolution en fréquence très faible. Ainsi, il y a également une sorte de compromis amplitude-fréquence à trouver.

Rectangulaire	**Bartlett (triangulaire)**	**Hamming**	**Hann**	**Blackman**
36 %	19 %	18 %	15 %	12 %

Tableau 11.1. *Imprécision maximale en amplitude pour quelques fenêtres classiques*

11.2.3. *Imprécision en phase*

La phase instantanée du partiel est parasitée par la phase de la fenêtre d'analyse (qui varie linéairement, modulo 2π, en fonction de la fréquence). La contribution de la phase de la fenêtre d'analyse n'est nulle que si la fréquence instantanée du partiel est une des fréquences de la transformée de Fourier discrète (multiples de $\mathcal{B}$), ce qui est rarement le cas.

Heureusement, il existe une technique de fenêtrage dite *zero-phase* qui annule, dans tous les cas, la contribution en phase de la fenêtre d'analyse. Quand cette technique est employée, la phase est constante pour tout le lobe principal (voir figure 11.4) et on obtient bien la phase du partiel au centre de la fenêtre temporelle en calculant $\arg(X[m])$, m étant l'indice du maximum local correspondant au partiel.

Pour cela, il faut considérer une fenêtre d'analyse de taille impaire, symétrique autour de la valeur centrale (souvent 1). Typiquement, aucune des valeurs de la fenêtre

n'est nulle. Une fenêtre de taille impaire peut alors poser problème, puisqu'une taille puissance de 2 est souvent nécessaire pour pouvoir utiliser efficacement un algorithme de transformée de Fourier rapide (FFT). La technique du *zero-padding* (voir paragraphe 11.1.3) peut alors être avantageusement employée pour centrer la fenêtre de taille impaire dans une fenêtre de taille puissance de 2 plus large, en complétant les échantillons manquants par des 0.

Notons au passage le cas singulier des fenêtres trigonométriques (Hann, Hamming et Blackman), naturellement N-périodiques. Leur valeur nulle en 0 peut être vue comme un *zero-padding* de taille 1 (voir section 11.1) qui n'a que peu d'influence sur le spectre, N étant en général très supérieur à 1.

La figure 11.6 illustre alors la transformation à faire subir au segment $w \times x$ avant la transformée de Fourier discrète.

11.2.4. *Choix de la fenêtre d'analyse*

Le choix de la fenêtre d'analyse est très important. Une discussion exhaustive sur les fenêtres d'analyse pourrait à elle seule faire l'objet d'un chapitre entier (voir [HAR 78]). Le tableau 11.2 présente un bref résumé des caractéristiques des fenêtres d'analyse les plus couramment employées. Des connaissances de base sur les fenêtres d'analyse sont nécessaires pour la bonne compréhension de la discussion qui suit. Des informations sur les fenêtres d'analyse peuvent être trouvées dans la littérature [HAR 78, OPP 89].

Les équations [11.30], [11.31], [11.32], [11.33] et [11.34] sont les équations des fenêtres d'analyse les plus communément utilisées : les fenêtres rectangulaire, de Bartlett, de Hamming, de Hann et de Blackman (versions N-périodiques pour les équations). Ces équations ne sont valables que pour $0 \leqslant n < N$. En dehors de cet intervalle leur valeur est 0 :

$$w_{\text{rectangulaire},N}(n) = 1 \qquad [11.30]$$

$$w_{\text{Bartlett},N}(n) = \begin{cases} \dfrac{2n}{N} & \text{si } \ 0 \leqslant n \leqslant \dfrac{N}{2} \\ 2 - \dfrac{2n}{N} & \text{si } \ \dfrac{N}{2} < n < N \end{cases} \qquad [11.31]$$

$$w_{\text{Hamming},N}(n) = 0{,}54 - 0{,}46\cos\Big(\frac{2\pi n}{N}\Big) \qquad [11.32]$$

$$w_{\text{Hann},N}(n) = \frac{1}{2}\Big(1 - \cos\Big(\frac{2\pi n}{N}\Big)\Big) \qquad [11.33]$$

$$w_{\text{Blackman},N}(n) = 0{,}42 - 0{,}5\cos\Big(\frac{2\pi n}{N}\Big) + 0{,}08\cos\Big(\frac{4\pi n}{N}\Big) \qquad [11.34]$$

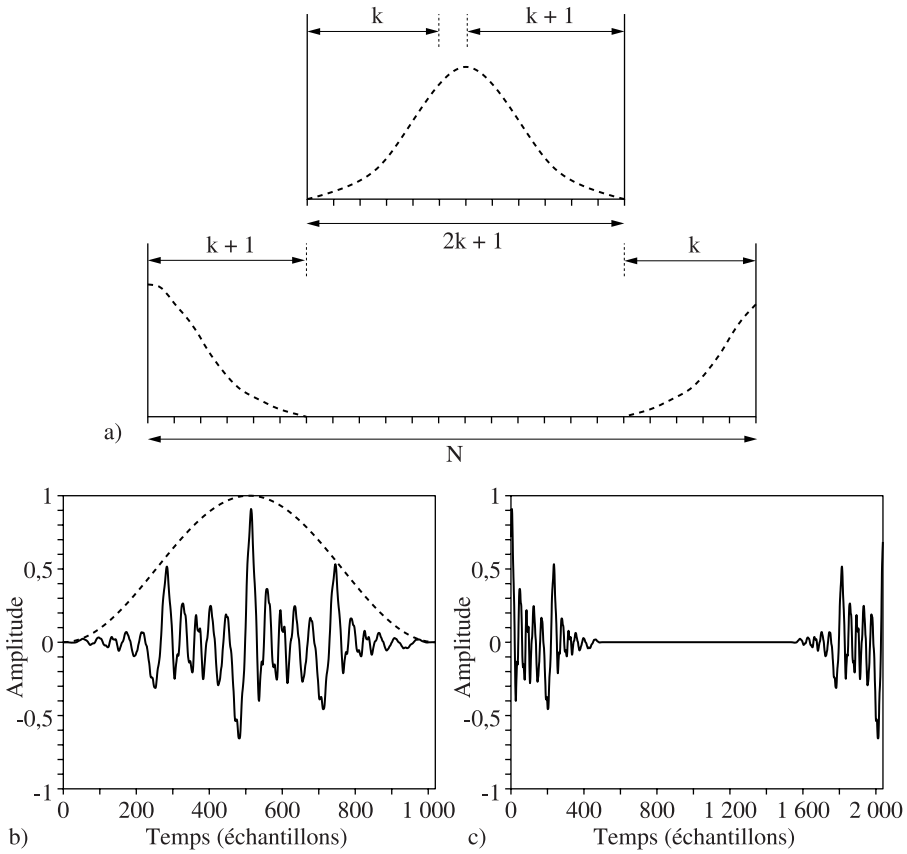


Figure 11.6. *La méthode de fenêtrage dite* zero-phase *(b) comparée à la méthode classique (phase linéaire, en a). La méthode* zero-phase *consiste à utiliser une fenêtre d'analyse de taille impaire (largeur $2k + 1$) centrée dans une fenêtre plus grande de taille N, en complétant les échantillons manquants par des zéros (technique du* zero-padding*). La fenêtre d'origine, de largeur $2k + 1$ et symétrique, doit être découpée en deux morceaux : le premier de taille k doit être placé en fin de la fenêtre de taille N tandis que le second de taille $k + 1$ doit être placé au début. Cette permutation (qui est en fait une translation si l'on considère la fenêtre de taille N comme étant périodique) est également illustrée dans le cas d'un signal plus complexe (c).*

La première est appelée la fenêtre rectangulaire à cause de sa forme dans le domaine temporel, qui est un rectangle. La figure 11.7 montre les représentations temporelles des autres fenêtres. La fenêtre de Bartlett est également appelée fenêtre triangulaire, toujours à cause de sa forme dans le domaine temporel. La fenêtre de Hann tient son nom de Julius von Hann, un météorologue autrichien, et apparaît souvent dans la littérature sous le nom de fenêtre de Hanning. En fait, le terme *hanning*

fut utilisé dans [BLA 58] par Blackman et Tukey pour décrire l'opération consistant à appliquer cette fenêtre à un signal et a depuis constitué le nom le plus communément utilisé pour cette fenêtre. Toutefois, ce terme *hanning* était initialement utilisé dans un contexte ironique, la fenêtre de Hann étant proche de la fenêtre de Hamming dans le domaine fréquentiel. Cependant, la fenêtre de Hann possède des propriétés très intéressantes et s'est révélée être très utile pour l'analyse et la synthèse sonore (voir aussi le chapitre 2). C'est pourquoi nous pensons plus juste de renoncer au sobriquet ironique de *hanning* pour ces chapitres. La figure 11.8 montre les spectres d'amplitude de ces fenêtres d'analyse. Remarquons au passage que même si les fenêtres de Hann et de Blackman paraissent se ressembler dans le domaine temporel, leurs spectres sont en fait très différents et ces fenêtres possèdent donc des propriétés différentes. Il existe bien évidemment beaucoup d'autres types de fenêtres, comme l'intéressante famille des fenêtres de Kaiser [OPP 89] qui permettent de choisir un compromis entre la largeur du lobe principal et le rapport entre l'amplitude de ce lobe et celles des lobes secondaires.

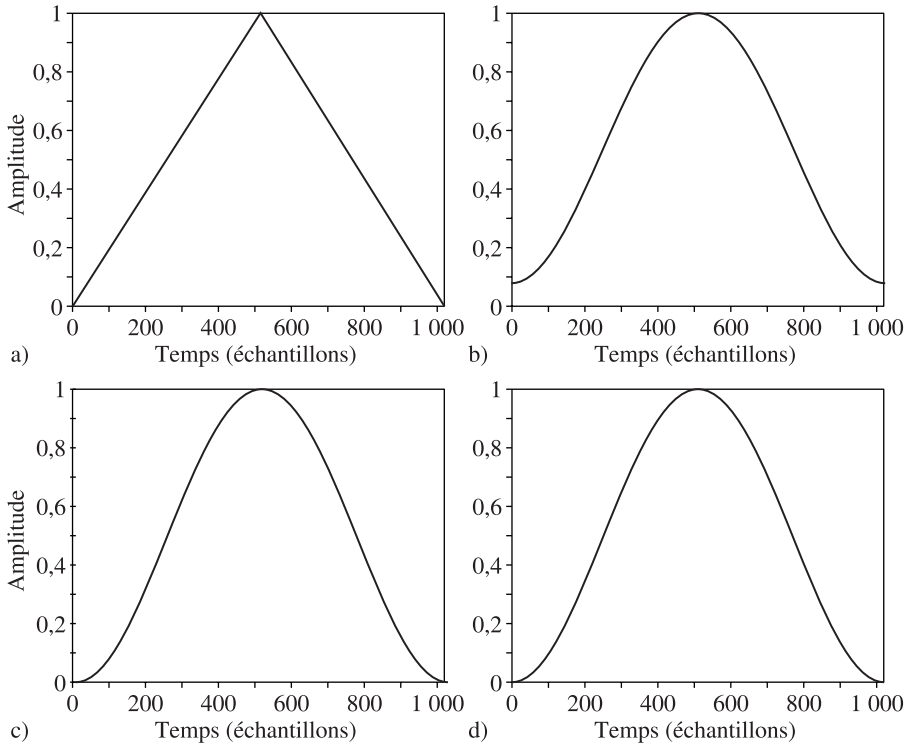


Figure 11.7. *Représentations temporelles de quelques fenêtres d'analyse parmi les plus classiques*

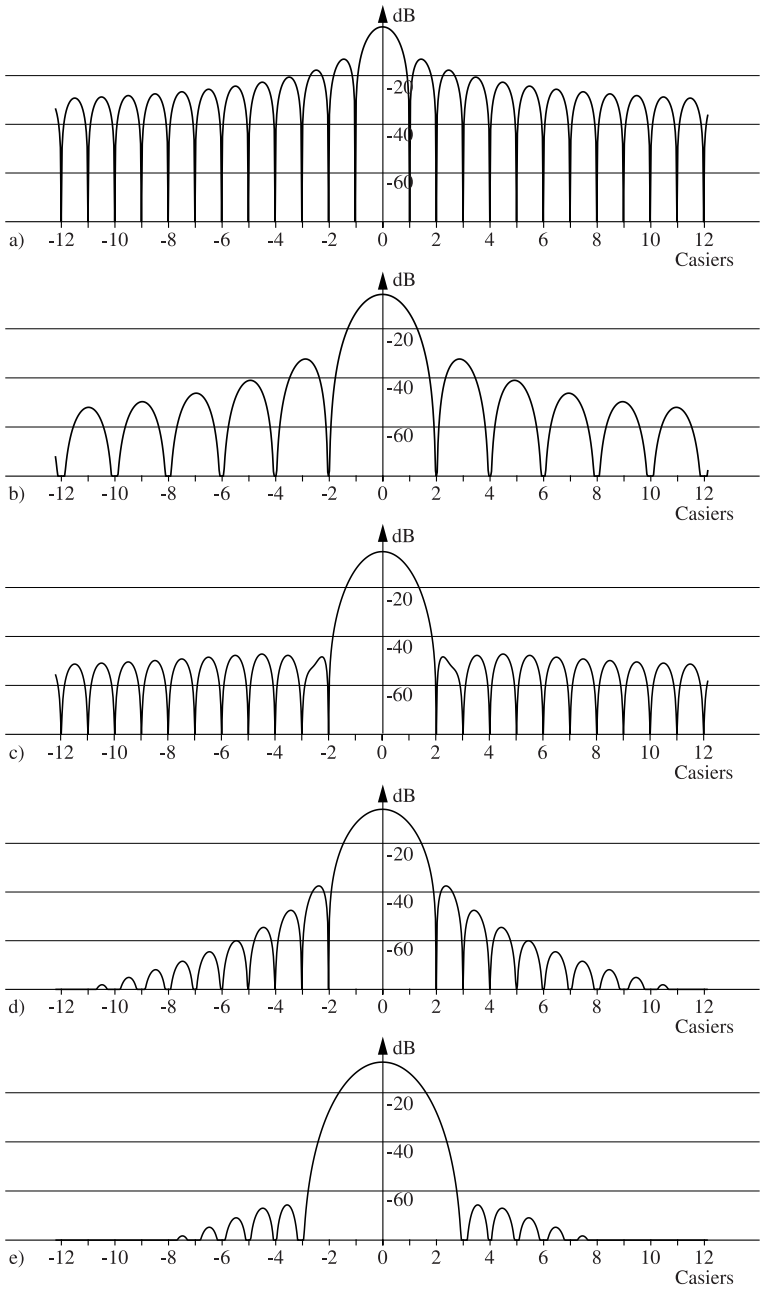


Figure 11.8. *Spectres d'amplitude de quelques fenêtres d'analyse, de haut en bas :* $W_{rectangulaire}$, $W_{Bartlett}$, $W_{Hamming}$, W_{Hann} *et* $W_{Blackman}$

11.2.4.1. *Type de la fenêtre*

Il convient tout d'abord de choisir le type de la fenêtre d'analyse, en observant trois critères.

Premièrement, un signal purement sinusoïdal devrait toujours produire un unique maximum local dans le spectre d'amplitude, afin que le nombre de maximums locaux corresponde au nombre de partiels du son. Plus formellement, avec $\mathcal{B} = \frac{F_s}{N}$, pour toute fréquence $0 \leqslant f < \mathcal{B}$, la suite $(|W(f + k\mathcal{B})|)_{k \geqslant 0}$ devrait être décroissante. Parmi toutes les fenêtres considérées dans le tableau 11.2, seules les fenêtres rectangulaire et de Hann possèdent cette propriété de « décroissance discrète ». En fait, ce critère n'est important que pour l'extraction et le suivi des partiels. Cette contrainte peut être relâchée en ne considérant que les maximums dont l'amplitude est au-dessus d'un certain seuil minimal. Les deux critères suivants sont plus importants car ils empêchent que des composantes fréquentielles voisines puissent interférer avec la mesure d'un partiel particulier.

Le deuxième critère est que les lobes secondaires doivent pouvoir être négligés en comparaison du lobe principal. C'est le cas pour la fenêtre de Hann, mais pas pour la fenêtre rectangulaire.

Enfin, le troisième critère veut que le lobe principal soit étroit, afin d'avoir une bonne résolution fréquentielle. Mais le prix à payer pour cela est d'augmenter les amplitudes des lobes secondaires. La fenêtre de Hann se révèle être un bon compromis. Le logiciel d'analyse *InSpect* [MAR 99, MAR 00] manipule de nombreuses fenêtres d'analyse et l'excellent comportement de la fenêtre de Hann a pu être vérifié en pratique.

Type	« **Décroissance discrète** »	**Lobes secondaires/principal**	**Largeur du lobe**
Rectangulaire	oui	-13 dB	$2\mathcal{B}$
Bartlett (triangulaire)	non	-26 dB	$4\mathcal{B}$
Hann	oui	-31 dB	$4\mathcal{B}$
Hamming	non	-42 dB	$4\mathcal{B}$
Blackman	non	-58 dB	$6\mathcal{B}$

Tableau 11.2. *Caractéristiques de quelques fenêtres d'analyse. Le ratio lobes secondaires/lobe principal est l'amplitude relative en dB entre le lobe principal et le plus fort lobe secondaire. La largeur du lobe principal est définie comme étant la distance symétrique entre les passages à 0 dans le domaine fréquentiel, de part et d'autre de l'origine.*

11.2.4.2. *Taille de la fenêtre*

La taille de la fenêtre d'analyse doit alors être précisée. D'une part, N doit être suffisamment large pour que la condition $\mathcal{B} < \min_{i \neq j}(|f_j - f_i|)$ soit vérifiée, ce qui

est toujours possible du fait des restrictions imposées en début de chapitre. La raison derrière cette condition est que deux fréquences doivent se trouver dans deux casiers de la transformée de Fourier différents, car $\frac{f_i a_i + f_j a_j}{a_i + a_j} \neq f_i + f_j$ et ceci causerait de graves problèmes dans l'équation [11.55] (le casier est alors dit « contaminé »). D'autre part, la fenêtre d'analyse doit rester suffisamment petite pour que les fréquences et les amplitudes n'aient pas le temps de trop varier durant l'intervalle de temps correspondant.

Une analyse spectrale à court terme précise est extrêmement importante pour beaucoup de méthodes d'analyse/synthèse utilisées en informatique musicale, comme la méthode de George et Smith [GEO 90] ou celle de McAulay et Quatieri [MCA 86], où il convient d'identifier avec précision les composantes fréquentielles (les partiels).

Lorsqu'aucune précaution n'est prise, les imprécisions énoncées précédemment apparaissent. Par conséquent, de nombreux logiciels d'analyse/synthèse souffrent des limitations de l'analyse de Fourier à court terme classique. De nombreuses techniques ont été proposées pour réduire ces limitations.

En dépit de ses nombreux inconvénients, la transformée de Fourier à court terme est souvent utilisée comme la première étape des méthodes d'analyse plus complexes. L'imprécision en fréquence Δ_f de l'analyse de Fourier à court terme (ou analyse de Gabor) est constante, inversement proportionnelle à une autre constante : la taille N de la fenêtre d'analyse.

Les ondelettes permettent d'obtenir une décomposition fréquentielle avec un facteur de qualité Δ_f / f constant, c'est-à-dire une imprécision en fréquence Δ_f proportionnelle à la fréquence f. Cette résolution est plus proche du comportement du système auditif.

Le problème est que les paramètres obtenus par l'analyse en ondelettes sont difficiles à interpréter et ne semblent pas du tout adaptés aux transformations musicales. Les harmoniques des sons complexes doivent être identifiées pour permettre des transformations musicales sans artefacts dramatiquement audibles. Un important problème survient alors avec le facteur de qualité Δ_f / f constant des transformées en ondelettes : pour les hautes fréquences, la résolution en fréquence est si mauvaise que plusieurs harmoniques se trouvent codées à l'intérieur d'un seul coefficient.

La suite de ce chapitre décrit des extensions de la transformée de Fourier qui ne présentent pas ce défaut et qui permettent d'avoir une meilleure précision à tout point de vue, même pour les basses fréquences (point faible de l'analyse de Fourier classique).

11.3. Méthodes pour l'amélioration de la précision

Nous souhaitons extraire les partiels du son, c'est-à-dire mesurer pour chacun d'entre eux, à chaque instant, les valeurs instantanées de sa fréquence, son amplitude et éventuellement sa phase. La suite de ce chapitre s'intéresse à des méthodes qui permettent de réduire les imprécisions en fréquence et en amplitude de l'analyse de Fourier classique, le problème de la phase ayant déjà été réglé au paragraphe 11.2.3.

Les partiels correspondent à des raies spectrales qui devraient *a priori* coïncider avec les maximums locaux dans le spectre d'amplitude à court terme. Pour déterminer les paramètres (fréquence, amplitude et phase) de chaque partiel, l'analyse de Fourier classique ne considère que le casier de la transformée de Fourier discrète qui contient l'amplitude la plus forte. Ceci ne donne les paramètres exacts que si la fréquence du partiel est exactement une des fréquences utilisée par la transformée de Fourier discrète (fréquences f_k de l'équation [11.12]), ce qui est très rarement le cas en pratique.

Une première solution consiste à utiliser la technique du *zero-padding* (voir paragraphe 11.1.3) sur le segment de signal à analyser, afin d'obtenir une version interpolée du spectre où la position du maximum local est donnée plus précisément, aussi bien en fréquence qu'en amplitude. Le problème est qu'il faut en pratique ajouter beaucoup de zéros pour obtenir une précision suffisante ; la transformée de Fourier à effectuer est alors de grande taille et très coûteuse en temps de calcul.

11.3.1. *Technique de l'interpolation parabolique*

En fait, quand la fréquence du partiel à analyser tombe au milieu de deux fréquences de la transformée de Fourier discrète, la convolution de la raie spectrale correspondant au partiel avec le spectre de la fenêtre d'analyse est échantillonnée sur chaque casier (le casier d'indice k correspondant à la fréquence f_k de l'équation [11.12]) de la transformée de Fourier discrète. L'effet produit sur le spectre est qu'il y a bien un maximum local pour un certain indice du spectre discret résultant, mais cette fois les casiers voisins ont également des amplitudes significatives, ce qui se voit clairement sur la figure 11.9. Afin d'augmenter la précision de l'analyse de Fourier classique, il est alors astucieux de considérer non seulement le casier où se trouve le maximum local, mais aussi certains de ses casiers voisins.

Les estimateurs sont des algorithmes qui utilisent le casier principal et quelques-uns de ses voisins à gauche et à droite afin de raffiner la mesure des paramètres du partiel.

Mais, des composantes spectrales proches en fréquences interfèrent entre elles additivement, affectant mutuellement leurs spectres propres. Pour cette raison, il est préférable de faire toutes les mesures spectrales à proximité du maximum local correspondant au partiel, afin de maximiser l'influence de ce partiel et de minimiser celle

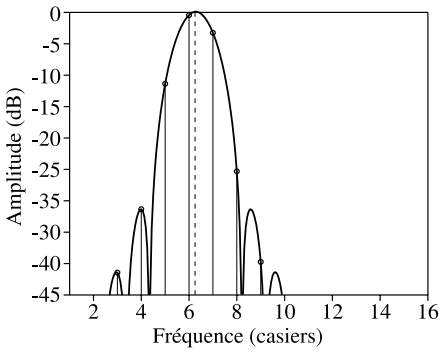


Figure 11.9. *Quand la fréquence analysée n'est pas un multiple de la plus petite des fréquences de la transformée de Fourier discrète ($\mathcal{B}$), les voisins du casier d'amplitude maximale ont également une amplitude significative. Les cercles ∘ sont les valeurs retrouvées à partir du spectre continu (trait plein) par la transformée de Fourier discrète.*

des pics adjacents. C'est pour cette raison qu'en pratique seuls les deux voisins immédiats à gauche et à droite sont utilisés, pour éviter que d'autres partiels qui seraient proches en fréquence n'interfèrent dans la mesure du partiel étudié.

En regardant de près la forme spectrale qui se dégage de l'amplitude du lobe principal de la plupart des fenêtres d'analyse, on constate qu'elle ressemble à une parabole, si toutefois l'échelle des décibels (dB) est utilisée pour l'amplitude. Ceci se vérifie sur la figure 11.9 et sur le zoom du lobe principal en figure 11.10. Pour chaque partiel, l'interpolation parabolique (aussi appelée interpolation quadratique) utilise le casier principal et ses voisins immédiats à gauche et à droite. Les coordonnées du sommet de la parabole passant par les trois points ainsi obtenus sont exactement la fréquence f_p et l'amplitude a_p cherchées pour le partiel p.

La méthode de Brent [PRE 92] est utilisée pour estimer le sommet de la parabole, ce qui donne l'algorithme suivant. Soit :

$$A_{\text{gauche}} = 20\log_{10}(|X[m_p - 1]|) \quad [11.35]$$

$$A_{\text{centre}} = 20\log_{10}(|X[m_p]|) \quad [11.36]$$

$$A_{\text{droite}} = 20\log_{10}(|X[m_p + 1]|) \quad [11.37]$$

La position du maximum local n'est connue que par l'indice entier du casier où il se trouve. Soit m_p cet indice. La formule suivante permet d'obtenir le décalage entre m_p et la position exacte du partiel p :

$$d = \frac{1}{2}\frac{A_{\text{gauche}} - A_{\text{droite}}}{A_{\text{gauche}} - 2A_{\text{centre}} + A_{\text{droite}}} \quad [11.38]$$

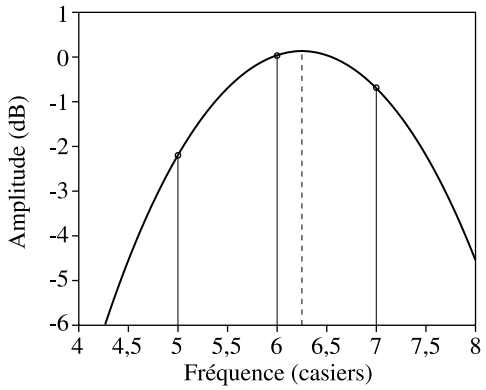


Figure 11.10. *Le maximum local et ses deux voisins immédiats se situent sur une parabole qui correspond à la forme du lobe principal de la fenêtre d'analyse. Les coordonnées du sommet de la parabole sont exactement la fréquence et l'amplitude cherchées*

Ce décalage d, également exprimé en casier, est un réel normalement compris dans l'intervalle $[-\frac{1}{2}; \frac{1}{2}]$. Il précise la position du partiel à l'intérieur du casier. La fréquence du partiel p cherchée est alors :

$$f_p = (m_p + d)\frac{F_s}{N} \qquad [11.39]$$

et l'amplitude en dB du partiel p est alors donnée par :

$$A_p = A_{\text{centre}} - (A_{\text{gauche}} - A_{\text{droite}})\frac{d}{4} \qquad [11.40]$$

d'où son amplitude en échelle linéaire :

$$a_p = 10^{A_p/20} \qquad [11.41]$$

Cette méthode donne de meilleurs résultats avec une fenêtre d'analyse gaussienne, puisque la forme de son lobe principal (son unique lobe d'ailleurs) est exactement une parabole si l'on considère son spectre avec l'échelle des décibels (dB) pour les amplitudes (voir figure 11.11). En effet, une gaussienne est de la forme $e^{\beta x^2}$, le spectre d'une gaussienne est une gaussienne et, en échelle logarithmique, on obtient pour le spectre d'amplitude une fonction du genre γx^2, qui correspond bien à une parabole. Mais la gaussienne est une fonction de support infini. En pratique, on ne peut générer que des gaussiennes tronquées de taille N, en utilisant par exemple l'équation suivante :

$$w_{\text{Gauss},\alpha,N}(n) = e^{-\alpha(\frac{2n}{N}-1)^2} \quad \text{pour} \quad 0 \leqslant n < N \qquad [11.42]$$

Il existe quantité d'estimateurs (voir [DON 99, JAC 00, KOO 99, QUI 94]). Citons simplement ici un autre estimateur tirant également parti de la forme du spectre de la fenêtre d'analyse. L'algorithme « du triangle » [ALT 99, KEI 01] tire son nom de la forme triangulaire, dans le domaine fréquentiel, de la fenêtre d'analyse utilisée. Keiler et Zölzer proposent d'utiliser cette fenêtre dont le spectre d'amplitude se résume à une fonction simple. Une forme triangulaire peut être décrite par seulement deux segments de droites, et une seule pente dans le cas considéré ici d'un triangle isocèle (puisque la fenêtre est symétrique). Il s'agit alors, pour chaque partiel, de retrouver les coordonnées du sommet du triangle qui lui correspond, ce qui est un peu plus difficile que dans le cas de la parabole vu précédemment.

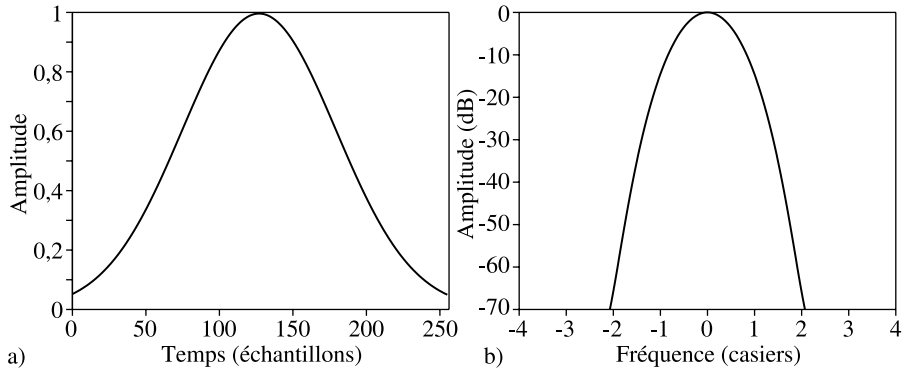


Figure 11.11. *La fenêtre d'analyse gaussienne tronquée générée par l'équation [11.42] avec $N = 256$ et $\alpha = 3$ (a) et son spectre d'amplitude (b), présentant un lobe principal de forme quasi parabolique*

11.3.2. *Utilisation des dérivées du signal*

Nous proposons dans [DES 00, MAR 98] la méthode de la transformée de Fourier à l'ordre n (FT^n). Cette méthode étend la transformée de Fourier à court terme classique en considérant également les n premières dérivées mathématiques du signal s. Elle tire parti de cette information supplémentaire pour améliorer la précision de l'analyse de Fourier non seulement en fréquence et en amplitude, mais aussi du point de vue de la résolution temporelle, ce qui repousse le compromis temps-fréquence de la transformée de Fourier classique. En effet, pour une même précision en fréquence, cette méthode se contente de fenêtres d'analyse plus petites.

Dans la suite de ce chapitre, nous nous intéressons uniquement au cas où $n = 1$ et nous ne considérons que la première dérivée du signal s.

11.3.2.1. *La théorie : le cas continu*

La position, la vitesse, mais aussi l'accélération, sont des paramètres clefs en cinématique. Il y a une réelle analogie entre ces paramètres et, respectivement, la phase ϕ_p, la fréquence $f_p = \frac{d\phi_p}{dt}$ et aussi la dérivée de la fréquence $\frac{df_p}{dt} = \frac{d^2\phi_p}{dt^2}$. Dans les modèles sinusoïdaux considérés (voir paragraphe 11.1.4), les fréquences f_p et les amplitudes a_p sont des paramètres qui varient lentement dans le temps. Ils peuvent être considérés comme quasi stationnaires durant une fenêtre d'analyse donnée et, par conséquent, $\frac{df_p}{dt}$ et $\frac{da_p}{dt}$ sont presque nulles. Dans ces conditions, l'utilisation de la dérivée du signal s peut aider à améliorer grandement la précision de l'analyse de Fourier aussi bien en fréquence qu'en amplitude. L'idée derrière cette méthode est extrêmement simple : la dérivée d'un sinus est elle-même un sinus, de phase différente, certes, mais de même fréquence.

11.3.2.1.1. Dérivée continue

Intéressons-nous maintenant à l'expression de la dérivée du signal s. Dans un modèle sinusoïdal, l'expression du signal est de la forme (voir les équations [11.20], [11.21] et [11.22]) :

$$s(t) = \sum_{p=1}^{P} \mathrm{osc}\big(f_p(t), a_p(t), \phi_p(0)\big)$$

Considérons un seul oscillateur :

$$o_p(t) = \mathrm{osc}\big(f_p(t), a_p(t), \phi_p(0)\big) = a_p(t)\cos\big(\phi_p(t)\big)$$

et calculons sa première dérivée :

$$\begin{aligned}\frac{do_p}{dt}(t) &= \frac{d}{dt}\big(a_p(t)\cos\big(\phi_p(t)\big)\big) \\ &= a_p(t)\,\frac{d}{dt}\big(\cos\big(\phi_p(t)\big)\big) + \frac{da_p}{dt}(t)\cos\big(\phi_p(t)\big)\end{aligned}$$

Nous ne nous intéressons qu'à des pics spectraux avec une amplitude significative, c'est-à-dire au-dessus d'une certain seuil minimal. C'est la raison pour laquelle nous ne considérerons pas le cas où le premier terme de la somme n'est pas significatif, puisqu'alors c'est l'ensemble de la somme qui est négligeable.

Comme a_p varie lentement dans le temps, on suppose que $\frac{da_p}{dt} \simeq 0$ et, quand le premier terme de la somme précédente a une valeur significative :

$$\frac{do_p}{dt}(t) \simeq a_p(t)\,\frac{d}{dt}\big(\cos\big(\phi_p(t)\big)\big) = -a_p(t)\,\frac{d\phi_p}{dt}(t)\sin\big(\phi_p(t)\big)$$

Mais on a $\frac{d\phi_p}{dt}(t) = 2\pi f_p(t)$ (d'après l'équation [11.22]) et donc :

$$\frac{do_p}{dt}(t) = -a_p(t)\left(2\pi f_p(t)\right)\sin\left(\phi_p(t)\right) = 2\pi\, a_p(t)\, f_p(t)\cos\left(\phi_p(t) - \frac{\pi}{2}\right) \quad [11.43]$$

Finalement, puisque dériver est une opération linéaire et d'après l'équation [11.20], la dérivée du signal s est :

$$\frac{ds}{dt}(t) = \sum_{p=1}^{P} 2\pi f_p(t) a_p(t)\cos\left(\phi_p(t) - \frac{\pi}{2}\right) \quad [11.44]$$

Notons FT^k la transformée de Fourier de la k-ième dérivée du signal $\frac{d^k s}{dt^k}$ ($k \geqslant 0$) et $|\mathrm{FT}^k(f)|$ la valeur de son amplitude à la fréquence f (et comme $\frac{d^0 s}{dt} = s$, la transformée de Fourier classique coïncide avec la transformée de Fourier d'ordre 0, FT^0).

L'intuition derrière l'équation [11.44] est que, pour chaque partiel p, il devrait y avoir un maximum (au même endroit) dans chaque spectre d'amplitude $|\mathrm{FT}^k|$ ($k \geqslant 0$). L'idée sous-jacente est alors d'utiliser les spectres d'amplitude $|\mathrm{FT}^k|$ pour différentes valeurs de k afin de déterminer les fréquences exactes de l'ensemble des partiels.

11.3.2.1.2. Déterminer la fréquence

Une conséquence de l'équation [11.44] est que, pour chaque partiel p, il y a un maximum local dans tous les $|\mathrm{FT}^k|$, à la fréquence f_p (en supposant naturellement que $a_p > 0$ et $f_p > 0$), et :

$$f_p = \frac{1}{2\pi}\frac{|\mathrm{FT}^1(f_p)|}{|\mathrm{FT}^0(f_p)|} \quad [11.45]$$

Bien que cette équation puisse paraître d'un intérêt très limité, puisqu'elle nécessite *a priori* la connaissance de f_p, sa version discrète se révèle en fait être d'un grand intérêt pratique. En effet, la discrétisation des spectres FT^k conduit à une valeur approchée f_p^0 de la fréquence f_p. La valeur exacte de cette fréquence peut alors être retrouvée à partir de cette valeur approchée en utilisant la version discrète de l'équation [11.45].

Il est très important de noter que les effets de la fenêtre d'analyse sont les mêmes sur $\mathrm{FT}^0(f_p)$ et sur $\mathrm{FT}^1(f_p)$, à la condition toutefois que la même fenêtre d'analyse soit appliquée pour calculer ces deux spectres. Ces effets se compensent grâce à la division dans l'équation [11.45].

11.3.2.1.3. Corriger l'amplitude

Bien entendu, une fenêtre d'analyse w est nécessaire pour obtenir un maximum local précis (un pic spectral étroit) dans le spectre $|\mathrm{FT}^k|$. La section 11.2 a présenté les conséquences du fenêtrage sur les spectres. Toutefois, l'amplitude exacte a_p du partiel p peut être facilement retrouvée à partir de sa valeur approchée $a_p^0 = |\mathrm{FT}^0(f_p)|$ puisque :

$$a_p = \frac{a_p^0}{|W(0)|} \qquad [11.46]$$

où W est le spectre de la fenêtre d'analyse w.

11.3.2.2. *La pratique : le cas discret*

Intéressons-nous maintenant à la version discrète de la méthode FT^1 précédente et étudions son intérêt pratique pour l'extraction précise des fréquences et des amplitudes des partiels.

La transformée FT^1 nécessite la connaissance de la première dérivée du signal s. Nous allons montrer que, en dépit de l'absence de tout moyen analogique/électronique qui aurait peut-être pu enregistrer cette information en même temps que le signal lui-même, la version discrète de la méthode FT^1 précédente peut être implantée en pratique.

Mais si la dérivée du signal n'est pas enregistrée avec le signal, elle doit pouvoir être calculée à partir du signal numérique lui-même, après la phase d'échantillonnage. Le signal discret s est échantillonné uniformément avec une fréquence F_s. Une valeur typique pour F_s est 44 100 Hz, qui correspond à la qualité des disques compacts (CD) audio.

11.3.2.2.1. Dérivée discrète (approximation)

La première dérivée $\frac{ds}{dt}$, aussi notée s', est mathématiquement définie par :

$$s'(t) = \lim_{\epsilon \to 0} \frac{s(t+\epsilon) - s(t)}{\epsilon}$$

En fait, les (demi-) dérivées à gauche et à droite sont, respectivement, pour ϵ négatif ou positif :

$$s'_-(t) = \lim_{\epsilon \to 0_-} \frac{s(t+\epsilon) - s(t)}{\epsilon} = \lim_{\eta \to 0_+} \frac{s(t) - s(t-\eta)}{\eta} \quad (\eta = -\epsilon)$$

$$s'_+(t) = \lim_{\epsilon \to 0_+} \frac{s(t+\epsilon) - s(t)}{\epsilon}$$

Comme s est une fonction $\mathcal{C}^\infty$, ses dérivées à gauche et à droite sont égales, c'est-à-dire que l'on a :

$$s'_-(t) = s'_+(t) = s'(t)$$

En pratique, le signal s est discret, $s[k]$ représentant $s(kT_s)$ (voir section 11.1), et le plus petit $|\epsilon|$ non nul possible pour le calcul de la dérivée est la période d'échantillonnage $T_s = 1/F_s$ elle-même. Par conséquent, considérons les approximations suivantes :

$$s'_-[k] = (s[k] - s[k-1])F_s$$

et :

$$s'_+[k] = (s[k+1] - s[k])F_s$$

Cette fois, $s'_+[k] \neq s'_-[k]$ en général, mais :

$$s'_+[k] = (s[k+1] - s[k])F_s = s'_-[k+1]$$
$$s'_-[k] = (s[k] - s[k-1])F_s = s'_+[k-1]$$

et par conséquent, il s'agit des mêmes fonctions discrètes à une translation de un échantillon près (et l'impact de cette petite translation sur le spectre d'amplitude de la dérivée du signal, qui est la seule chose considérée par la méthode, est négligeable).

Prenons alors arbitrairement l'approximation de la dérivée à gauche comme approximation de la dérivée elle-même (en prenant $s[k] = 0$ pour $k < 0$) :

$$s'[k] = F_s(s[k] - s[k-1]) \qquad [11.47]$$

L'équation [11.47] a une forme tout à fait caractéristique en théorie des filtres numériques [MOO 90, SMI 85] ; plus précisément, elle définit un filtre passe-haut. Son équation est :

$$y[n] = F_s x[n] - F_s x[n-1] \qquad [11.48]$$

et sa fonction de transfert se calcule facilement :

$$H(z) = F_s(1 - z^{-1}) \qquad [11.49]$$

et finalement son gain est :

$$|H(e^{i\omega})| = F_s\sqrt{2\big(1 - \cos(\omega)\big)} = 2F_s \sin\Big(\frac{\omega}{2}\Big) \quad \text{où} \quad \omega = \frac{2\pi f}{F_s} \qquad [11.50]$$

Dériver est une opération linéaire qui peut aussi être considérée comme une opération de filtrage avec un gain $2\pi f$ (voir équation [11.43]), c'est-à-dire $F_s\omega$, ce qui

constitue un gain théorique bien différent de celui obtenu en pratique $|H(e^{i\omega})|$, tout particulièrement pour des valeurs de ω élevées. La figure 11.12 illustre ce phénomène. Fort heureusement, cette différence peut être corrigée en multipliant après coup le spectre d'amplitude de la « dérivée » (approchée par l'équation [11.47]) du signal par le facteur F défini dans l'équation suivante :

$$F(\omega) = \frac{\omega}{2\sin\left(\frac{\omega}{2}\right)} \quad \text{avec} \quad \omega = \frac{2\pi f}{F_s} \qquad [11.51]$$

Il est également facile de montrer que les pics spectraux sont bien conservés par un gain qui croît assez lentement avec la fréquence. La figure 11.13 illustre ce phénomène. Une autre possibilité, donnant de meilleurs résultats, est de retrouver la fréquence directement à partir du gain du filtre. Pour un pic spectral donné, si v_0 et v_1 sont les valeurs (strictement positives) mesurées respectivement dans le spectre d'amplitude du signal et dans celui de sa dérivée approchée, et ce au même endroit correspondant à la fréquence f, alors :

$$v_1 = 2F_s \sin\left(\frac{\omega}{2}\right) \cdot v_0$$

et par conséquent :

$$\omega = 2\arcsin\left(\frac{v_1}{2F_s v_0}\right)$$

c'est-à-dire :

$$f = \frac{1}{\pi F_s} \arcsin\left(\frac{v_1}{2F_s v_0}\right) \qquad [11.52]$$

Nous avons choisi ici d'approcher la dérivée par un filtre linéaire du premier ordre. En fait, la méthode pourrait être généralisée et quantité d'autres filtres pourraient aussi bien être utilisés à la place, puisque le changement d'amplitude du signal à une fréquence donnée pourrait être comparé avec le changement attendu en théorie pour le filtre. L'avantage principal d'utiliser la différence entre deux échantillons successifs est que l'on reste ainsi proche de la théorie (cas continu) tout en obtenant de bons résultats en pratique. Si, dans le futur, la dérivée du signal pouvait être enregistrée avec le signal lui-même, ceci rendrait inutile le calcul de la dérivée approchée.

Notons DFT^k la transformée de Fourier discrète de la k-ième dérivée du signal ($k \geqslant 0$), sur N échantillons consécutifs à partir d'un certain endroit l, et $|\text{DFT}^k[m]|$ la valeur du spectre d'amplitude pour le casier numéro m. Plus formellement :

$$|\text{DFT}^k[m]| = \frac{2}{N}\left|\sum_{n=0}^{N-1} \frac{d^k a}{dt^k}[l+n]\, w[n]\, e^{-j\frac{2\pi}{N}nm}\right|$$

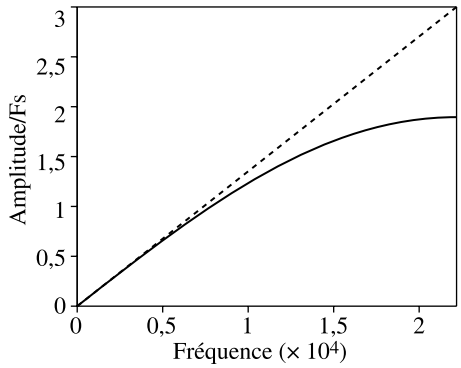


Figure 11.12. *Gain en pratique $|H|$ (trait plein) comparé au gain théorique de la dérivation (pointillés)*

Pour un partiel p de fréquence f_p et d'amplitude a_p, l'analyse de Fourier classique donne les valeurs approchées suivantes pour ces paramètres :

$$f_p^0 = m_p \frac{F_s}{N} \quad [11.53]$$

$$a_p^0 = |\text{DFT}^0[m_p]| \quad [11.54]$$

où m_p est l'indice où se trouve le maximum local dans $|\text{DFT}^0|$ correspondant à la fréquence f_p. En considérant également $|\text{DFT}^1|$, des valeurs bien plus précises peuvent être obtenues pour la fréquence et l'amplitude.

Une conséquence des équations [11.20] et [11.43] est qu'il y a, pour chaque partiel p, un maximum local à l'indice m_p dans les deux spectres d'amplitude $|\text{DFT}^0|$ et $|\text{DFT}^1|$, comme illustré en figure 11.14.

11.3.2.2.2. Fréquence précise

Un équivalent discret de l'équation [11.45] est :

$$f_p = \frac{1}{2\pi} \frac{|\text{DFT}^1[m_p]|}{|\text{DFT}^0[m_p]|} \quad [11.55]$$

Plus précisément, m_p est l'entier le plus proche de $f_p \frac{N}{F_s}$, $m_p = \lfloor f_p \frac{N}{F_s} + \frac{1}{2} \rfloor$ et $\mathcal{B}\left(m_p - \frac{1}{2}\right) \leqslant f_p < \mathcal{B}\left(m_p + \frac{1}{2}\right)$. Si ce n'était pas le cas pour la fréquence f_p calculée, alors l'analyse à l'ordre 1 aurait échoué pour cette fréquence, ce qui serait une indication que le casier était en fait occupé (« contaminé ») par plusieurs composantes fréquentielles.

La fréquence exacte de chaque partiel peut être déterminée en considérant l'équation [11.55] pour chaque maximum m dans $|\text{DFT}^0|$.

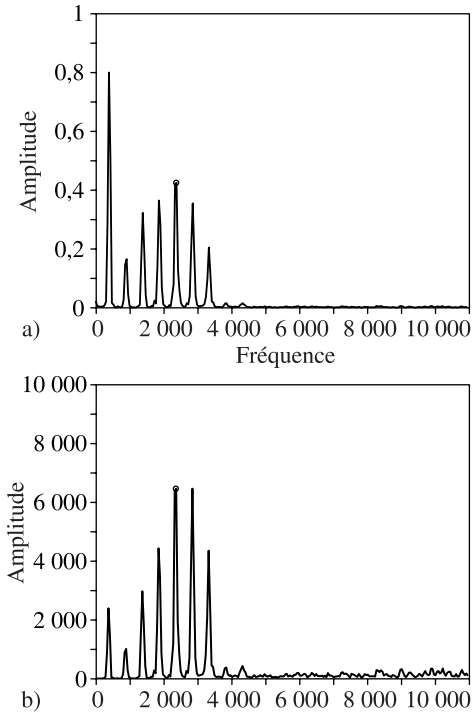


Figure 11.13. *Spectres d'amplitude de DFT*^{0} *et DFT*^{1}

Encore une fois, comme dans le cas continu, il est extrêmement important de noter que les effets de la fenêtre d'analyse sont les mêmes à la fois sur $|\text{DFT}^0[m_p]| = v_0$ et $|\text{DFT}^1[m_p]| = v_1$, à condition naturellement d'utiliser la même fenêtre d'analyse pour calculer ces deux spectres. Ces effets se compensent toujours grâce à la division dans l'équation [11.55].

11.3.2.2.3. Amplitude précise

L'amplitude exacte de chaque partiel p peut alors être déterminée, à partir des valeurs approchées a_p^0 de l'amplitude et f_p^0 de la fréquence, à l'aide de la fréquence précise f_p calculée auparavant, en utilisant la version discrète de l'équation [11.46] :

$$a_p = \frac{a_p^0}{|W(|f_p - f_p^0|)|} \qquad [11.56]$$

Cette équation nécessite également la connaissance du spectre W (sa version continue) de la fenêtre d'analyse, comme ceux présentés en figure 11.8.

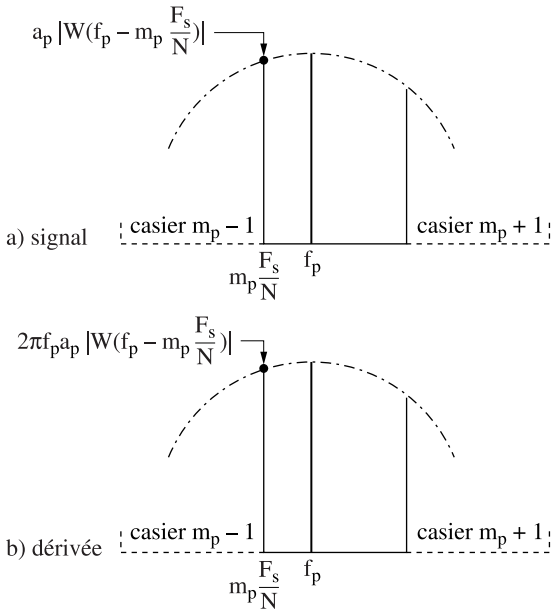


Figure 11.14. *Bien que la fréquence f_p à analyser se trouve au milieu de la plage de fréquence correspondant au m_p-ième casier de la transformée de Fourier, elle produit un pic au même endroit dans les spectres du signal (a) et de sa dérivée (b), les amplitudes correspondantes étant bien évidemment différentes.*

Il peut être précalculé dans une table, en utilisant une technique de *zero-padding* (voir paragraphe 11.1.3) pour en avoir une version discrète suréchantillonnée approchant la version continue.

Toutefois, le spectre W de la fenêtre d'analyse peut souvent être calculé analytiquement. C'est le cas pour la fenêtre de Hann.

Considérons tout d'abord la fenêtre rectangulaire de taille N :

$$w_{\text{rectangulaire},N}(n) = \begin{cases} 1 & \text{si } \ 0 \leqslant n < N \\ 0 & \text{sinon} \end{cases}$$

et calculons sa transformée de Fourier en temps discret en utilisant l'équation [11.7] :

$$W_{\text{rectangulaire},N}(\omega) = \sum_{n=-\infty}^{+\infty} w_{\text{rectangulaire},N}[n]\, e^{-i\omega n}$$

Comme la fenêtre rectangulaire, de taille N, est de support fini :

$$W_{\text{rectangulaire},N}(\omega) = \sum_{n=0}^{N-1} e^{-i\omega n}$$

Dans le cas où $\omega = 0$, $W_{\text{rectangulaire},N}(0) = N$.

Si $\omega \neq 0$, alors on reconnaît la somme des N premiers termes d'une suite géométrique de raison $e^{-i\omega}$ et de premier terme 1, ainsi :

$$W_{\text{rectangulaire},N}(\omega) = \frac{1 - e^{-iN\omega}}{1 - e^{-i\omega}}$$

que l'on peut aussi écrire sous la forme :

$$W_{\text{rectangulaire},N}(\omega) = \frac{e^{-iN\omega/2}}{e^{-i\omega/2}} \cdot \frac{e^{+iN\omega/2} - e^{-iN\omega/2}}{2i} \cdot \frac{2i}{e^{+i\omega/2} - e^{-i\omega/2}}$$

et finalement, en reconnaissant dans ce qui précède l'expression de deux sinus :

$$W_{\text{rectangulaire},N}(\omega) = \begin{cases} e^{-i(N-1)\omega/2} \cdot \dfrac{\sin(N\omega/2)}{\sin(\omega/2)} & \text{si } \omega \neq 0 \\ N & \text{sinon} \end{cases} \qquad [11.57]$$

En utilisant le fait que :

$$\cos(2\pi n/N) = \frac{e^{+i2\pi n/N} + e^{-i2\pi n/N}}{2}$$

et que, cette expression s'annulant pour $n = N$, la fenêtre de Hann N-périodique :

$$w_{\text{Hann},N}(n) = \frac{1}{2}\Big(1 - \cos\Big(\frac{2\pi n}{N}\Big)\Big) \quad \text{pour} \quad 0 \leqslant n < N \quad \text{(et 0 sinon)}$$

peut aussi s'écrire sous la forme :

$$\begin{aligned} w_{\text{Hann},N}(n) = \frac{1}{2} w_{\text{rectangulaire},N+1}(n) &- \frac{1}{4} e^{+i2\pi n/N} w_{\text{rectangulaire},N+1}(n) \\ &- \frac{1}{4} e^{-i2\pi n/N} w_{\text{rectangulaire},N+1}(n) \end{aligned}$$

On peut alors facilement vérifier (en utilisant le fait que $e^{i\pi} = e^{-i\pi} = -1$) que :

$$\begin{aligned} W_{\text{Hann},N}(\omega) = \frac{1}{2} W_{\text{rectangulaire},N+1}(\omega) &\\ + \frac{1}{4} W_{\text{rectangulaire},N+1}\Big(\omega + \frac{2\pi}{N}\Big) &\\ + \frac{1}{4} W_{\text{rectangulaire},N+1}\Big(\omega - \frac{2\pi}{N}\Big) & \end{aligned} \qquad [11.58]$$

Il est alors facile (en factorisant $e^{-iN\omega/2}$, de module 1) d'obtenir l'expression analytique de $|W_{\text{Hann},N}(\omega)|$:

$$|W_{\text{Hann},N}(\omega)| = \frac{1}{2}\left|\frac{\sin(\omega\frac{N+1}{2})}{\sin(\omega\frac{1}{2})} + \frac{1}{2}\frac{\sin\big((\omega+\frac{2\pi}{N})\frac{N+1}{2}\big)}{\sin\big((\omega+\frac{2\pi}{N})\frac{1}{2}\big)} + \frac{1}{2}\frac{\sin\big((\omega-\frac{2\pi}{N})\frac{N+1}{2}\big)}{\sin\big((\omega-\frac{2\pi}{N})\frac{1}{2}\big)}\right| \quad [11.59]$$

$$\text{si } \omega \neq 0 \text{ et } N \text{ sinon}$$

d'où l'on tire immédiatement l'expression analytique de $|W_{\text{Hann},N}(f)|$ utilisée dans l'équation [11.56], puisque d'après l'équation [11.9], on a :

$$f = \frac{\omega F_s}{2\pi}$$

On a donc $W_{\text{Hann},N}(0) = N$. Toutefois, en pratique, il faut normaliser $|W|$ ainsi calculé en le divisant par la taille de la transformée de Fourier N (ceci est dû au fait qu'aucun facteur de normalisation n'a été appliqué par la transformée de Fourier en temps discret (équation [11.7]) utilisée pour calculer analytiquement $|W|$, or on suppose que l'amplitude a_p^0 de l'équation [11.56] a été, elle, normalisée).

Cette méthode d'analyse, appelée aussi parfois « algorithme de la dérivée », est l'une des méthodes les plus précises à ce jour [KEI 02].

Du point de vue de la complexité, cette méthode est très intéressante puisqu'elle nécessite le calcul de deux transformées de Fourier de petite taille au lieu d'une transformée de Fourier bien plus grande. En utilisant la transformée de Fourier rapide (FFT), sa complexité est la même que pour l'analyse de Fourier classique, c'est-à-dire $O(N\log(N))$. Mais, en pratique $N = 256$ est souvent suffisant pour $F_s = 44\,100$ Hz avec la méthode DFT[1], alors que l'analyse de Fourier classique nécessite des valeurs de N bien plus grandes (environ huit fois plus grandes en pratique).

11.4. Bibliographie

[ALL 77] ALLEN J.B., « Short-term spectral analysis, synthesis, and modification by the discrete Fourier transform », *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, n° 3, p. 235-238, 1977.

[ALT 99] ALTHOFF R., KEILER F., ZÖLZER U., « Extracting sinusoids from harmonic signals », dans *Proceedings of the Second COST G-6 Workshop on Digital Audio Effects (DAFx 99)*, NTNU, Trondheim, p. 97-100, décembre 1999.

[ARF 91] ARFIB D., « Analysis, transformation, and resynthesis of musical sounds with the help of a time-frequency representation », dans *Representation of Musical Signals*, chapitre 3, p. 87-118, MIT Press, Cambridge, Massachusetts, 1991.

[BLA 58] BLACKMAN R.B., TUKEY J.W., *The Measurement of Power Spectra*, Dover Publications, New York, 1958.

[COO 65] COOLEY J.W., TUKEY J.W., « An algorithm for the machine computation of complex Fourier series », *Mathematics of Computation*, vol. 19, p. 297-301, avril 1965.

[DES 00] DESAINTE-CATHERINE M., MARCHAND S., « High precision Fourier analysis of sounds using signal derivatives », *Journal of the Audio Engineering Society*, vol. 48, n° 7/8, p. 654-667, juillet/août 2000.

[DOL 86] DOLSON M.B., « The phase vocoder: A tutorial », *Computer Music Journal*, vol. 10, n° 4, p. 14-27, 1986.

[DON 99] DONADIO M., « How to interpolate the peak location of a DFT or FFT if the frequency of interest is between bins », http://www.dspguru.org/howto/tech/peakfft.htm, 1999.

[FIT 96] FITZ K., HAKEN L., « Sinusoidal modeling and manipulation using Lemur », *Computer Music Journal*, vol. 20, n° 4, p. 44-59, 1996.

[FRI 03] FRIGO M., JOHNSON S.G., FFTW User's Manual, MIT, http://www.fftw.org, 2003.

[GAB 46] GABOR D., « Theory of communication », *Journal of the Institute for Electrical Engineers*, vol. 93, p. 429-457, 1946 (partie III).

[GEO 90] GEORGE E.B., SMITH M.J.T., « Analysis-by-synthesis/overlap-add sinusoidal modeling applied to the analysis and synthesis of musical tones », *Journal of the Audio Engineering Society*, vol. 40, n° 6, p. 497-516, juin 1990.

[HAR 78] HARRIS F.J., « On the use of windows for harmonic analysis with the discrete Fourier transform », dans *Proceedings of the IEEE*, vol. 66, p. 51-83, janvier 1978.

[IRC 96] IRCAM, Paris, AudioSculpt user's manual, seconde édition, avril 1996.

[JAC 00] JACOBSEN E., « Frequency estimation page », http://www.primenet.com/~ericj/fe.htm, 2000.

[KAI 80] KAISER J.F., SCHAFER R.W., « On the use of the I_0-sinh window for spectrum analysis », *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, n° 1, p. 105-107, février 1980.

[KEI 01] KEILER F., ZÖLZER U., « Extracting sinusoids from harmonic signals », *Journal of New Music Research*, vol. 30, n° 3, p. 243-258, septembre 2001.

[KEI 02] KEILER F., MARCHAND S., « Survey on extraction of sinusoids in stationary sounds », dans *Digital Audio Effects (DAFx) Conference*, Hambourg, Allemagne, p. 51-58, septembre 2002.

[KOO 99] KOOTSOOKOS P.J., « Some frequency estimation algorithms », http://www.clubi.ie/PeterK/freqalgs.html, 1999.

[LAG 02] LAGRANGE M., MARCHAND S., RAULT J.-B., « Sinusoidal parameter extraction and component selection in a non stationary model », dans *Digital Audio Effects (DAFx) Conference*, Hambourg, Allemagne, p. 59-64, septembre 2002.

[MAR 86] MARQUES J., ALMEIDA L., « A background for sinusoid based representation of the voiced speech », dans *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Tokyo, Japon, p. 1233-1236, 1986.

[MAR 98] MARCHAND S., « Improving spectral analysis precision with an enhanced phase vocoder using signal derivatives », dans *Digital Audio Effects (DAFx) Conference*, Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research), Barcelone, Espagne, p. 114-118, novembre 1998.

[MAR 99] MARCHAND S., STRANDH R., « InSpect and ReSpect: Spectral modeling, analysis and real-time synthesis software tools for researchers and composers », dans *International Computer Music Conference (ICMC)*, International Computer Music Association (ICMA), Pékin, Chine, p. 341-344, octobre 1999.

[MAR 00] MARCHAND S., « InSpect software package », http://www.scrime.u-bordeaux.fr/InSpect, 2000.

[MAS 95] MASRI P., BATEMAN A., « Identification of nonstationary audio signals using the FFT, with application to analysis-based synthesis of sound », dans *IEE Colloquium on Audio Engineering*, The Institution of Electrical Engineers (IEE), Londres, Grande-Bretagne, p. 11.1-11.6, 1995.

[MAS 98] MASRI P., CANAGARAJAH N., « Extracting more detail from the spectrum with phase distortion analysis », dans *Digital Audio Effects (DAFx) Conference*, Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research), Barcelone, Espagne, p. 119-122, novembre 1998.

[MCA 86] MCAULAY R.J., QUATIERI T.F., « Speech analysis/synthesis based on a sinusoidal representation », *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, n° 4, p. 744-754, 1986.

[MOO 78a] MOORE F.R., « An introduction to the mathematics of digital signal processing », *Computer Music Journal*, vol. 2, n° 1, p. 38-47, 1978 (partie 1).

[MOO 78b] MOORE F.R., « An introduction to the mathematics of digital signal processing », *Computer Music Journal*, vol. 2, n° 2, p. 48-60, 1978 (partie 2).

[MOO 85] MOORE F.R., « An introduction to the mathematics of digital signal processing », dans *Digital Audio Signal Processing*, p. 1-67, A-R Editions, Inc, Madison, Wisconsin, The Computer Music and Digital Audio Series, 1985.

[MOO 90] MOORE F.R., *Elements of Computer Music*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[MOOR 77] MOORER J.A., « Signal processing aspects of computer music–A survey », *Computer Music Journal*, vol. 1, n° 1, p. 4-37, 1977.

[MOOR 78] MOORER J.A., « The use of the phase vocoder in computer music applications », *Journal of the Audio Engineering Society*, vol. 26, n° 1/2, p. 42-45, 1978.

[MOOR 85] MOORER J.A., « Signal processing aspects of computer music: A survey », dans *Digital Audio Signal Processing*, p. 149-220, A-R Editions, Inc, Madison, Wisconsin, The Computer Music and Digital Audio Series, 1985.

[NYQ 28] NYQUIST H., « Certain topics in telegraph transmission theory », *AIEE Transactions*, p. 617-644, 1928.

[OPP 89] OPPENHEIM A.V., SCHAFER R.W., *Discrete-time signal processing*, Prentice-Hall, Englewood Cliffs, New Jersey, Signal Processing Series, 1989.

[ORF 96] ORFANIDIS S.J., *Introduction to Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, Signal Processing Series, 1996.

[POR 80] PORTNOFF M.R., « Time-frequency representation of digital signals and systems based on short-time Fourier transform », *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, n° 8, p. 55-69, février 1980.

[PRE 92] PRESS W.H., TEUKOLSKY S.A., VETTERLING W.T., FLANNERY B.P., « Minimization or maximization of functions », dans *Numerical Recipes in C (The Art of Scientific Computing)*, chapitre 10, p. 402-405, Cambridge University Press, Cambridge, seconde édition, 1992.

[QUI 94] QUINN B.G., « Estimating frequency by interpolation using Fourier coefficients », *IEEE Transactions on Signal Processing*, vol. 42, n° 5, p. 1264-1268, mai 1994.

[SER 97] SERRA M.-H., « Introducing the phase vocoder », dans *Musical signal processing*, chapitre 2, p. 31-90, Swets et Zeitlinger, Lisse, Pays-Bas, Studies on New Music Research, 1997.

[SERR 89] SERRA X., A system for sound analysis/transformation/synthesis based on a deterministic plus stochastic decomposition, Thèse de doctorat, CCRMA, Département de musique, Université de Stanford, 1989.

[SERR 90] SERRA X., SMITH J.O., « Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition », *Computer Music Journal*, vol. 14, n° 4, p. 12-24, 1990.

[SERR 97] SERRA X., « Musical sound modeling with sinusoids plus noise », dans *Musical Signal Processing*, chapitre 3, p. 91-122, Swets et Zeitlinger, Lisse, Pays-Bas, Studies on New Music Research, 1997.

[SHA 49] SHANNON C.E., « Communication in the presence of noise », dans *Proceedings of IRE*, p. 10-12, janvier 1949.

[SMI 85] SMITH J.O., « An introduction to digital filter theory », dans *Digital Audio Signal Processing*, p. 69-135, A-R Editions, Inc, Madison, Wisconsin, The Computer Music and Digital Audio Series, 1985.

[VER 98a] VERMA T.S., MENG T.H.Y., « An analysis/synthesis tool for transient signals that allows a flexible sines+transients+noise model for audio », dans *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, IEEE, 1998.

[VER 98b] VERMA T.S., MENG T.H.Y., « Time scale modification using a sines+transients+noise signal model », dans *Digital Audio Effects (DAFx) Conference*, Audiovisual Institute, Pompeu Fabra University and COST (European Cooperation in the Field of Scientific and Technical Research), Barcelone, Espagne, p. 49-52, novembre 1998.

Chapitre 12

MPEG, une norme pour la compression, la structuration et la description du son

12.1. Introduction

Les standards de compression audio MPEG forment une famille de formats et d'outils, visant à répondre à une multitude de besoins, qui vont de la diffusion de radio et télévision numérique au stockage et à la diffusion sur les réseaux informatiques, en particulier Internet.

Le groupe MPEG (*Motion Picture Expert Group*) appartient à un sous-comité (SC 29) de l'ISO-IEC (*International Standards Organization-International Electrotechnical Commission*). Il lui appartient de définir les standards des formats de transmission d'audio et vidéo numérique à faibles débits. A ce jour, quatre standards ont été officiellement consacrés : MPEG-1 en 1992 [MPEG 92], MPEG-2 en 1994 [MPEG 94], MPEG-4 en 1998 [MPEG 98] et MPEG-7 en 2002 [MPEG 02]. Un nouveau standard est en cours d'élaboration. Chaque standard MPEG définit en outre différents sous-ensembles de fonctionnalités (*layers*, *profiles* et *levels*), qui compliquent encore un peu plus la situation, parce qu'ils correspondent à autant de variantes et de cas typiques d'utilisation de la norme. Dans le domaine de l'audio numérique, on peut distinguer les deux premiers standards, qui concernent la compression perceptuelle des signaux sonores échantillonnés, des deux suivants, qui concernent la structuration (MPEG-4) et la description (MPEG-7) des scènes sonores. Une vue d'ensemble des standards et des formats MPEG est présentée en figure 12.1.

Chapitre rédigé par Rémi RONFARD.

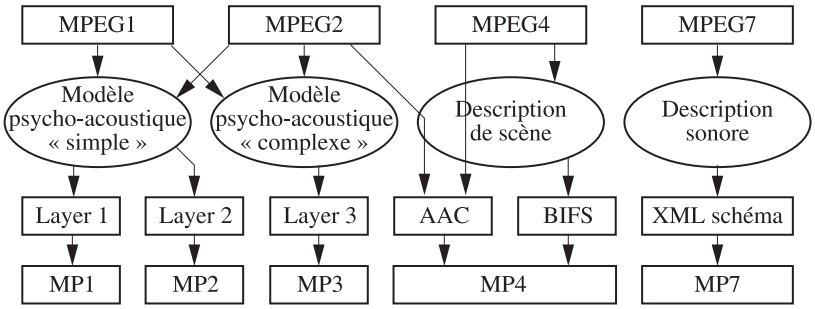


Figure 12.1. *Formats et standards MPEG Audio*

Pour aider le lecteur à s'y retrouver, il est bon de revenir dans un premier temps sur les bases algorithmiques et psycho-acoustiques de la compression MPEG audio (section 12.2), avant de détailler les différents formats, qui correspondent aux layers 1, 2 et 3 de MPEG-1 et MPEG-2 (les fameux formats MP1, MP2 et MP3). Ensuite, nous poserons la question des rapports entre la compression et les techniques de traitement et d'analyse. Quel est le rôle de ces standards dans le cadre de l'analyse du signal musical ? D'une part, ils fournissent des techniques et des outils de compression et des modèles psycho-acoustiques qui peuvent également servir à l'analyse. D'autre part, il devient de plus en plus fréquent que la source à analyser soit donnée sous l'un des formats de compression MPEG, ce qui pose la question de l'analyse des signaux compressés, qu'il s'agisse de sons naturels (section 12.4) ou synthétiques (section 12.5). Enfin, la norme MPEG-7 introduit un format générique pour l'organisation et la description des scènes sonores (section 12.6), destiné à unifier les voies qui permettent de passer du signal au signe musical.

12.2. Bases psycho-acoustiques et algorithmiques

La compression MPEG audio est basée sur un modèle de la perception auditive qui permet de prédire les différences perceptibles entre un signal sonore non compressé et le même signal après compression.

Cette prédiction permet de minimiser les différences perceptibles occasionnées par la compression. Il s'agit d'une compression avec perte, qualifiée de transparente puisque même un auditeur averti ne peut percevoir les dégradations, en vertu du modèle psycho-acoustique. L'évaluation d'une technique de compression transparente est subjective, et le choix des solutions retenues dans les standards MPEG résulte de tests comparatifs.

12.2.1. *Quantification et compression*

A la base de toutes les techniques de compression utilisées dans MPEG, il y a la notion de quantification, qui fait passer d'une grandeur continue à une grandeur quantifiée. Pour bien comprendre l'importance de cette étape, on peut représenter le signal quantifié comme la somme d'un signal original x_n et d'un bruit de quantification ε_n :

$$x_n \xrightarrow{quantification} Qx_n = x_n + \varepsilon_n$$

Le principe de base de la compression va être de faire en sorte que le signal compressé Qx_n masque le bruit de quantification, de sorte que l'on n'entende que le signal original, tout en maintenant un niveau de quantification le plus faible possible. Cette idée simple donne lieu à toute une série de techniques de compressions, dites « perceptuelles », qui tirent parti des particularités psycho-acoustiques de l'oreille humaine.

Dans le cas du disque compact, le signal audio est échantillonné 44 100 fois par seconde et quantifié avec une résolution uniforme de 16 bits. Ce codage en *pulse code modulation* (PCM) nécessite donc 705 600 bits par seconde. On peut tirer parti des propriétés psycho-acoustiques de l'oreille en choisissant une quantification non uniforme comme la fameuse loi mu [GER 92].

Dans les techniques plus récentes comme MPEG, le signal est décomposé temporellement en trames et fréquentiellement en sous-bandes, comme on le verra plus loin. La quantification est variable et non uniforme, pour chaque trame et chaque sous-bande afin d'utiliser au mieux le débit disponible, en vertu d'un modèle psycho-acoustique plus riche.

12.2.2. *Psycho-acoustique*

Dans la perception de sons complexes comme ceux de la télévision ou de la musique entre en jeu un grand nombre de phénomènes perceptifs et cognitifs, qui permettent de reconnaître et de localiser les sources sonores. Ces phénomènes sont commandés par certaines propriétés particulières de l'oreille qui permettent de comprendre comment sont construites les perceptions de hauteur et d'intensité sonores, comment les sons sont individualisés ou fusionnés, et comment un son peut en masquer un autre. On pourra se reporter à [MOO 89] pour une présentation plus complète des phénomènes complexes liés à la perception du son.

L'*intensité* d'un son (*sound pressure level*) se mesure en variations de pression par référence à un niveau de référence, généralement selon une échelle logarithmique : les décibels. L'intensité perçue (*loudness*) est une fonction complexe du SPL, dont l'unité est le phone, et qui peut être mesurée expérimentalement, dans des conditions normales d'audition. Il n'existe pas de théorie complète de la perception de l'intensité : ainsi, un bruit de moteur continue d'être perçu comme « bruyant » même à grande distance, alors que son intensité physique est faible. Cependant, la connaissance des courbes expérimentales de l'intensité perçue en fonction du temps et de la fréquence permet de comprendre certains mécanismes auditifs, dont la compression peut ensuite tirer parti.

Au centre des mécanismes de perception, il y a la cochlée, une sorte de tube enroulé en spirale, dont la fréquence de résonance varie en fonction de la longueur selon une loi approximativement logarithmique, de 20 Hz au sommet (apex) à 20 kHz à la base. La résolution temporelle de la cochlée est d'environ 200 ms, et l'intensité perçue continue donc à croître dans cette période. La résolution fréquentielle est en moyenne précise à 1/12 de ton, ce qui démontre une sélectivité en fréquence de l'oreille (facteur Q) excellente ; elle dépend du niveau sonore (elle augmente avec les SPL).

En dépit de son excellente discrimination des hauteurs, la cochlée présente une réponse fréquentielle complexe, large et asymétrique. La largeur du signal d'excitation de la cochlée, appelée également largeur de bande critique, est de l'ordre de la centaine de hertz pour les basses fréquences, et atteint environ un tiers d'octave dans les hautes fréquences. La largeur des bandes critiques permet d'expliquer un phénomène crucial pour la compression audio : l'effet de masque.

Par définition, une bande critique est la bande de fréquence à l'intérieur de laquelle les intensités sonores sont additives, du point de vue de la perception. Cette largeur varie avec la fréquence, et peut être mesurée expérimentalement. Elle donne lieu à une échelle non uniforme de fréquences, appelée échelle Bark. Par définition, un Bark est l'intervalle de fréquence séparant deux bandes critiques. On peut observer la réponse en fréquence de la cochlée en faisant varier de manière continue la différence de fréquence entre deux sons purs ; on perçoit alors des battements jusqu'à une différence de 15 Hz environ, puis un son unique et dissonant jusqu'à une différence d'environ un tiers d'octave (100 à 1 000 Hz), puis enfin deux sons séparés et consonants. Ce deuxième seuil correspond à la largeur de bande critique.

L'effet de masque joue principalement à l'intérieur d'une même bande critique. Il dépend du caractère tonal ou bruité des sons. Dans le modèle psycho-acoustique de MPEG, on considère qu'un son tonal masque jusqu'à 18 dB en dessous de son niveau sonore, et qu'un son bruité masque seulement 6 dB. Autrement dit, le bruit de quantification à l'intérieur d'une bande critique est inaudible jusqu'à 6 dB en

dessous d'un son bruité, et jusqu'à 18 dB en dessous d'un son tonal. La compression MPEG tire également parti des effets de masque interbandes qui sont prédits par le calcul des fonctions d'étalement (*spreading functions*).

L'effet de masque joue également temporellement, en vertu de la réponse temporelle de la cochlée : un son masque le bruit de quantification jusqu'à 30 ms avant et 200 ms après lui. L'utilisation de fenêtres d'analyse courtes, de l'ordre de 30 ms, permet d'utiliser le masquage temporel dans la compression MP3 pour corriger les effets de pré-échos.

Les modèles psycho-acoustiques utilisés en MPEG ont été élaborés pour des sons monophoniques. Lorsque le son est spatialisé (stéréo, *surround*), des phénomènes plus complexes apparaissent, comme l'hypersensibilité et la précédence. En présence de plusieurs sources sonores localisées, l'oreille peut adapter ses seuils de perception en fonction de la direction (*cocktail party effect*) et développer ainsi une hypersensibilité non prévue par les modèles psycho-acoustiques simples. Par ailleurs, bien que les intensités s'additionnent pendant 30 ms, l'oreille peut localiser les sources sonores dans l'espace en tenant compte de la direction des premiers sons perçus, si les artefacts de compression ne lui donnent pas d'indications erronées. En conséquence, la qualité subjective de la compression peut être dégradée pour des enregistrements stéréo ou *surround* de grande qualité, sur ces enceintes de grande qualité également.

12.2.3. *Décomposition en sous-bandes*

Les techniques de compression MPEG audio effectuent une décomposition du signal en trames temporelles et en bandes fréquentielles appelées « sous-bandes », afin de prédire localement et aussi précisément que possible les effets de masque et les niveaux de quantification utiles.

Le principe consiste à décomposer le signal en plusieurs signaux de largeur de bande réduite sous-échantillonnés, et de coder chaque sous-bande de manière économique, en exploitant les effets de masquage prédits par le modèle psycho-acoustique. La dynamique de chaque sous-bande est optimisée (« compandée ») par le calcul d'un gain de sous-bande, et les échantillons de chaque sous-bande sont requantifiés avec une longueur de mot de sous-bande spécifique.

Le calcul des sous-bandes est un compromis entre la fidélité au modèle psycho-acoustique (les bandes critiques), l'efficacité des calculs de coefficients de sous-bandes, et la qualité de l'algorithme de reconstruction. Le codage en sous-bandes des formats de compression MPEG audio utilise deux techniques de décomposition en sous-bandes, issues de formats plus anciens et développés indépendamment – le

MUSICAM et l'ASPEC. Issu des travaux du CCETT, de l'IRT et de Philips, le format de compression MUSICAM [DEH 91] a été adopté par MPEG pour les couches MP1 (en version simplifiée) et MP2 (en version complète). La couche MP3 provient pour sa part d'un compromis entre le MUSICAM et le format ASPEC développé par ATT, Thomson, l'institut Fraunhoffer et le CNET en France [BRA 91]. Le MUSICAM est basé sur une banque de filtres en miroirs quadratiques polyphasés (QMF), tandis que l'ASPEC est basé sur une transformée en cosinus modifiée (MDCT). On retrouve donc ces deux techniques dans les trois couches MPEG.

Dans la technique des banques de filtres en quadratures (QMF), la décomposition en sous-bandes est réalisée par une banque de trente-deux filtres polyphasés H_i :

$$S(t,i) = \sum_{n=0,511} x(t-n)\, H_i(n)$$

$$H_i(n) = h(n)\cos(2i+1)(n-16)\frac{\Pi}{64}$$

A partir de 512 échantillons, on reconstitue ainsi 32 sous-bandes S(t,i) i = 1 ... 32, qui peuvent être largement sous-échantillonnées. Un filtre inverse reconstitue le signal initial sans perte. Ces 32 sous-bandes sont une approximation très grossière des bandes critiques, puisque ces dernières sont non uniformes et en plus grand nombre [SWI 90]. En particulier, le faible nombre de sous-bandes et leur largeur excessive (environ 650 Hz) ne permettent pas de mettre en œuvre les modèles psycho-acoustiques les plus précis. La compression MPEG de layers 1 et 2, qui utilise cette technique, se contente donc d'un modèle psycho-acoustique simplifié.

Dans la compression de layer 3, une décomposition fréquentielle plus fine est mise en œuvre, grâce à la technique de transformation en cosinus modifié (*Modified Discrete Cosine Transform*). La MDCT permet de mettre en œuvre une décomposition en sous-bandes avec des fenêtres se recouvrant à 50 %, de telle manière que l'aliasage puisse être complètement compensé par le filtre de reconstruction. La MDCT est également utilisée par les formats de compression Dolby AC3 et MPEG AAC.

La particularité de cette technique est qu'il s'agit d'un filtre dont le nombre d'échantillons en temps et en fréquence sont égaux, malgré un recouvrement temporel de 50 %, ce qui permet une économie de codage. A titre de comparaison, une transformée de Fourrier glissante avec un recouvrement identique produit deux fois plus d'échantillons fréquentiels que d'échantillons temporels. Le résultat d'une MDCT consiste en M sous-bandes B_k de réponses impulsionnelles :

$$f_k(n) = h(n)\sqrt{\frac{2}{M}}\cos\left(n+\frac{M+1}{2}\right)\left(k+\frac{1}{2}\right)\frac{\prod}{M}$$

Comme dans le cas des filtres QMF, on obtient ainsi une décomposition de l'espace temps-fréquence, dans laquelle on va pouvoir projeter efficacement les résultats de l'analyse psycho-acoustique.

12.3. Les formats MPEG-1 et MPEG-2

Les standards MPEG-1 et MPEG-2 audio sont suffisamment similaires pour que l'on puisse les confondre dans la suite. La distinction la plus importante concerne les trois layers définis par MPEG-1 et repris par MPEG-2.

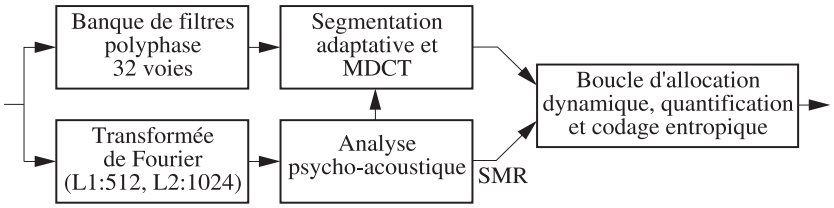


Figure 12.2. *Schéma général du codeur MP3, d'après Ted Painter [PAI 00]*

La structure d'ensemble d'un codeur MPEG-1 est décrite en figure 12.1 (layers 1 et 2) et figure 12.2 (layer 3). Dans chaque cas, les échantillons audio sont traités en parallèle selon deux chaînes d'analyse, l'une consacrée au codage proprement dit (filtrage, décomposition en sous-bandes, allocation et codage des trames), et l'autre à la prévision psycho-acoustique. Notons que les détails du modèle psycho-acoustique et de l'implémentation du codeur ne sont pas standardisés, bien que certains éléments soient suggérés, et que le standard comporte deux modèles psycho-acoustiques de référence. Le seul élément de validation (conformant) d'un codeur MPEG est donc sa capacité à générer un flux « standard », c'est-à-dire lisible par le décodeur MPEG standard, qui est lui entièrement spécifié par le standard.

En entrée, le décodeur MPEG-1 accepte des fréquences d'échantillonnage de 32, 44,1 et 48 kHz. En MPEG-1, on autorise les flux mono, dual channel ou stéréo (deux canaux codés indépendamment) et joint stéréo (deux canaux codés ensemble). Un mode à cinq canaux a été ajouté dans le MPEG-2. Les débits de sorties sont de 32, 48, 56, 64, 96, 112, 128, 192, 256 et 384 KB/S.

A la différence du codeur, le décodeur MPEG-1 est entièrement spécifié par le standard, ce qui permet le contrôle de validité des flux MPEG, puisqu'il n'existe qu'un unique décodeur (au moins pour chaque layer, profile ou level), et assure leur pérennité (puisqu'il est possible de construire un décodeur MPEG à la lecture du standard). La structure générale du décodeur est représentée en figure 12.3.

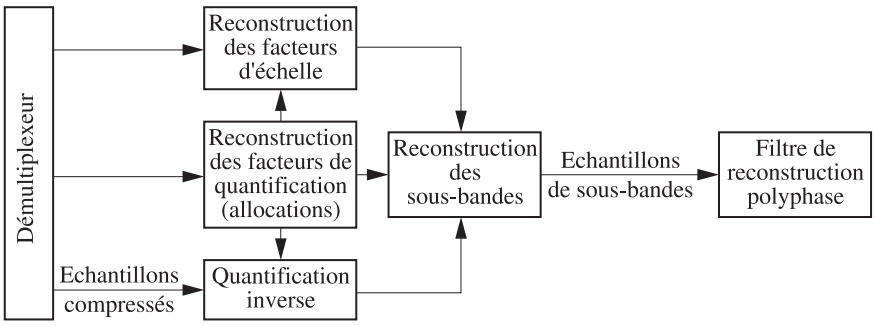


Figure 12.3. *Schéma général du décodeur MP1, d'après Watkinson [WAT 99]*

12.3.1. *MPEG-1 Layer 1*

Avant compression, le signal est lu par blocs de 384 échantillons PCM. C'est la trame audio, d'une durée de 8 ms à 48 kHz, 8,7 ms à 44,1 kHz et 12 ms à 32 kHz. On a donc une « fréquence de trame » de l'ordre de 83,3 à 125 Hz. Un filtre QMF polyphasé est appliqué pour calculer 12 échantillons dans chacune des 32 sous-bandes ($12 \times 32 = 384$). Chaque sous-bande est donc sous-échantillonnée par un facteur de 32, soit une fréquence de sous-bande de 1 000 à 1 500 Hz.

Dans chaque trame, le modèle psycho-acoustique est utilisé pour calculer un facteur d'échelle sur 6 bits (entre 0 et 64) et une longueur de mots sur 4 bits (les échantillons pouvant être codés sur 0 à 15 bits).

Chaque trame audio du *bitstream* mp1 est donc constituée ainsi :

– un code de synchronisation,

– une entête (*header*) contenant la fréquence d'échantillonnage et la valeur de pré-emphase,

– un code de synchronisation et de correction d'erreur (CRC) 32 codes d'allocation de 4 bits (*wordlengths*),

– 32 facteurs d'échelle de 6 bits (gains),

– 32×12 échantillons de sous-bandes, de taille variable (entre 0 et 15 bit chacun),

– des données annexes (*ancillary data*) non spécifiées par le standard.

Le décodeur MP1 lit les 384 échantillons dans des mots de 16 bits en les décalant selon les codes d'allocation, applique les facteurs d'échelle, et reconstitue le signal par un filtre de reconstruction (QMF polyphasé inverse), comme indiqué en figure 12.3.

Entête (32)	Code de correction d'erreur (CRC) (0,16)	Facteurs d'allocation de bits (128-256)	Facteurs d'échelle (0-384)	Echantillons de sous-bandes	Données annexes

Figure 12.4. *Format de trame audio MP1 (d'après Davis Pan [PAN 95])*

12.3.2. *MPEG-1 Layer 2*

La compression MPEG-1 de niveau 2 (MPE2) est une version assez complète de la technique MUSIC AM, dont le MPE1 est la version simplifiée. En MP2, les trames sont constituées de trois blocs de 384 échantillons, soit 1 152 échantillons par trame. La trame MP2 dure donc 24 ms à 48 kHz, 26 ms à 44,1 kHz et 36 ms à 32 kHz. Pour chaque trame, le spectre est estimé séparément par TFR à 1 024 points, afin d'affiner le modèle psycho-acoustique. Pour chaque bloc de 384 échantillons, un facteur d'échelle et un code d'allocation sont calculés, comme en MP1. La fréquence de trame est de l'ordre de 40 Hz.

Après compression, chaque trame MP2 contient trois blocs de 384 échantillons, codés comme en MP1 à l'exception des facteurs d'échelle (on peut trouver 1, 2 ou 3 facteurs d'échelle par sous-bande et par trame), comme indiqué en figure 12.4.

Entête (32)	Code de correction d'erreur (0,16)	Codes d'allocation (26-188)	SCFSI (0-60)	Facteurs d'échelle (0-1080)	Echantillons de sous-bandes	Données annexes

Figure 12.5. *Format de trame audio MP2 (d'après Davis Pan [PAN 95])*

12.3.3. *MPEG-1 Layer 3*

La compression MPEG-1 de niveau 3 (MPE3) combine une banque de 32 filtres QMF avec une transformation en cosinus (MDCT) à douze bandes, afin d'obtenir une décomposition temps-fréquence beaucoup plus fine qu'en MP1 ou MP2. Ceci permet la mise en œuvre d'un second modèle psycho-acoustique, plus riche qu'en MP1 et MP2.

Les deux étapes permettent d'obtenir encore 1 152 échantillons quantifiés de manière non uniforme et codés selon un code de Huffman. Les trames audio sont codées dans des chaînes de longueur fixe, mais de manière à remplir tout l'espace à l'aide d'un mécanisme de « réservoir » permettant de conserver les espaces non utilisés par une trame pour les trames suivantes (voir figure 12.5).

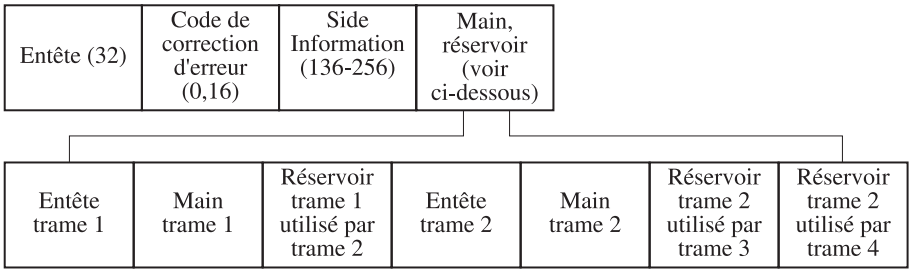


Figure 12.6. *Format de trame audio MP3 (d'après Davis Pan [PAN 95])*

Dans les niveaux 1 et 2, le débit de sortie était considéré comme fixe (*Constant Bit Rate* ou CBR), de sorte que le codeur devait déterminer les gains et les codes d'allocation permettant d'optimiser la qualité sonore, pour une taille de trame constante. Dans le niveau 3, le débit est variable d'une trame à l'autre (*Variable Bit Rate* ou VBR), dans des bornes fixées par le standard, grâce à un mécanisme de réservoirs qui permet de conserver une fraction de chaque trame pour les trames suivantes, lorsque le codage de la trame est plus efficace que nécessaire (voir encore figure 12.6).

12.3.4. *MPEG-2 Advanced Audio Coding*

Lors de la préparation de la norme MPEG-2, les trois layers de MPEG-1 ont été repris quasiment à l'identique, avec quelques extensions dont la plus importante concerne le son surround. Ce mode consiste en cinq canaux, avec une compatibilité ascendante qui permet la lecture des flux MPEG-2 surround par un lecteur MPEG-1 stéréo. Cependant, ceci s'est fait au détriment de la qualité de compression, et on a vu apparaître également une nouvelle série d'algorithmes qui n'assuraient plus la compatibilité avec le MPEG-1 et étendaient les techniques mises en œuvre dans le layer 3, sous le nom de MPEG-2 AAC (*advanced audio coding*) et NBC (*non backward compatible*) [BOS 97].

Les formats définis en MPEG-2 AAC concernent la compression haute-fidélité de l'audio numérique multipistes, avec un débit de 64 kbs pour un maximum de 48 pistes, et une qualité subjective supérieure de 50 % au MPEG-2 BC dans le cas du

son surround. Le MPEG-2 AAC est basé entièrement sur la transformée en cosinus (MDCT), sans banque de filtres QMF et s'apparente de ce point de vue au Dolby-AC3. Peu utilisées jusqu'à présent, les techniques AAC ont été reprises récemment dans le cadre de la norme MPEG-4.

12.4. Traitement et analyse dans le domaine de compression

Depuis l'apparition et la diffusion massive des formats de compression MPEG, plusieurs auteurs ont proposé d'utiliser le domaine de compression pour effectuer des traitements numériques sans introduire de nouvelles boucles de décompression-recompression, que ce soit pour ajouter des effets numériques [LEV 96] ou pour indexer les données par leur contenu [PAN 95, WAT 99].

12.4.1. *Traitements et effets*

Levine a décrit en 1996 une implémentation de quelques effets audio numériques dans le domaine de compression du MP2 : écho, flanger, chorus et réverbération [LEV 96], tirant parti de la relative indépendance entre les sous-bandes, qui permettent d'appliquer les effets dans chaque sous-bande. L'extension au MP3 et au MPEG-AAC se révèle beaucoup plus délicate, en raison de la complexité et de la finesse des sous-bandes obtenues par MDCT. En conséquence, une variante de ces techniques est proposée dans sa thèse au CCRMA de l'Université de Standford [LEV 96], afin de permettre simultanément compression et traitement.

12.4.2. *Analyse et indexation*

Une autre catégorie de traitements audio qui se développe aujourd'hui est l'extraction de *features*, dont l'objectif est d'indexer les segments sonores par leur contenu, afin de permettre à un moteur de recherche de retrouver ces segments dans une grande base, en réponse à une requête formulée par un ou plusieurs exemples.

L'utilisation par MPEG de modèles psycho-acoustiques suggère l'idée d'effectuer cette extraction de *features* dans le domaine compressé, afin de tirer parti de l'économie de représentation que l'on peut y trouver. De plus, cette approche présente l'avantage de gagner du temps en n'effectuant qu'un décodage partiel.

Une application caractéristique de ces méthodes est la classification de segments sonores en catégories telles que silence, parole, musique et autres bruits, sur la base de *features* extraits des échantillons de sous-bandes du flux MPEG [WAT 99]. La classification des segments audio a lieu à l'échelle du plan (de quelques secondes

à quelques minutes) par classification sur les statistiques de l'énergie à court terme, des fréquences fondamentales (*pitch*), des ratios d'énergies entre sous-bandes et du taux de pause (nombre de passage d'une trame de silence à une trame de signal), sur l'ensemble des trames d'un plan.

Tous ces features peuvent être approximés de manière efficace à partir des échantillons de sous-bande. Pour le calcul des énergies à court terme, il s'agit d'une simple somme des énergies de sous-bandes. Pour la détermination des fréquences fondamentales, les auteurs se limitent aux fréquences inférieures à 900 Hz, ce qui leur permet de restreindre la recherche à la première sous-bande, qui contient les fréquences jusqu'à 689 Hz pour un signal échantillonné à 41,1 kHz, et 750 Hz pour un signal échantillonné à 48 kHz. Comme dans le cas des effets numériques, ces approximations sont valables dans les bandes de fréquence larges des layers MP1 et MP2, mais s'appliquent beaucoup plus difficilement au MP3 ou au MPEG-AAC.

Notons que pour ces applications, il serait encore plus judicieux de disposer du modèle psycho-acoustique, qui fournit une représentation mieux adaptée aux problèmes posés. Ainsi, les intensités et les indices de tonalité calculés pour chaque bande critique offrent une meilleure représentation du signal sonore que la décomposition en sous-bandes. Mais ces données sont irrémédiablement perdues après compression – ce pourrait être une application de la future norme MPEG-7 que de permettre la transmission de telles données, qui ne sont pas strictement nécessaires au décodage, mais enrichissent la description des sons compressés.

12.5. MPEG-4 et les objets sonores

Du point de vue qui nous intéresse ici, les principales innovations du standard MPEG-4 audio concerne la description structurée de scènes composées de plusieurs objets audio, et la prise en compte de l'audio synthétique (événements midi, musique échantillonnée, etc.). Le point de vue proposé par MPEG-4 est donc de préserver la structure jusque dans la compression et la transmission de données. Les applications de ce standard sont différentes de celles des standards précédents, dans la mesure où il n'existe pas, à l'heure actuelle, de moyens de reconstituer la structure des scènes (sonores ou visuelles) à partir de donnés audio ou vidéo « traditionnelles ». Il n'y a donc ni continuité ni compatibilité entre les standards MPEG-1 et MPEG-2 d'une part, et MPEG-4 d'autre part, mais bien une rupture technologique, tournée vers les nouveaux moyens de production audio et vidéo, et qui anticipe de nouveaux modes de transport et de stockage. En conséquence, le décodeur MPEG-4 devient un outil de composition et de synthèse.

12.5.1. *Description de scènes audio (*MPEG-4 Audio BIFS*)*

BIFS (*Binary Format for Scenes*) est le format de composition de scènes du standard MPEG-4 [SCH 99]. Comme dans les standards précédents, il se décompose en une partie vidéo et une partie audio. Une scène BIFS est un graphe d'objets audio et vidéo, selon un modèle qui reprend pour une large part les concepts définis par VRML, mais sans qu'il y ait forcément référence à un espace 3D. Ainsi, la composition de nœuds audio dans le graphe de scène peut soit simuler une disposition spatiale des sources sonores dans un monde virtuel, comme en VRML, soit simplement être spécifiée comme une liste d'effets sonores à appliquer à chaque piste (comme dans une liste de mixage audio traditionnelle).

Dans une scène BIFS, le standard MPEG-4 définit trois grandes catégories d'objets audio :

– l'audio compressé à faible débit (*general audio*),

– la musique de synthèse (*synthetic audio*),

– la parole de synthèse (*speech*).

12.5.2. *MPEG-4 General Audio*

Le standard MPEG-4 couvre des débits allant de 2 à 64 kbs. Les formats de compression audio définis pour MPEG-4 reprennent l'intégralité des outils MPEG-AAC, et introduisent plusieurs nouveautés. Le codage de parole échantillonnée à 8 kHz est réalisé par des techniques paramétriques, avec des débits compris entre 2 et 6 kbs. Le codage de parole et de musique échantillonnés à 8 ou 16 kHz peut également être réalisé par une prédiction linéaire (*Code Excited Linear Prediction*) avec des débits compris entre 6 et 24 kbs. Pour les débits compris entre 24 et 64 kbs, les techniques de codage en fréquence de la norme MPEG-2 AAC ont été reprises, et une nouvelle technique de quantification vectorielle (TwinVQ) a été introduite. Remarquons l'abandon de la compatibilité avec les formats MPEG-1 et MPEG-2, et la prise en compte de nouvelles fonctionnalités telles que la possibilité de modifier le tempo sans changer la hauteur et *vice versa*, le codage explicite des effets numériques (réverbération, écho, flanger) et leur prise en compte par le décodeur MPEG-4, ou la transmission progressive qui permet de transmettre et de décoder des versions plus ou moins fidèles du signal, selon la bande passante disponible.

12.5.3. *MPEG-4 Structured Audio*

Les formats dédiés à la synthèse musicale ne sont pas des formats de compression à proprement parler, mais plutôt des formats permettant la transmission et le stockage des paramètres de synthèse afin de retarder la synthèse jusqu'au

décodeur MPEG-4. Ce modèle, qui s'inspire des langages de synthèse d'images comme PostScript ou RenderMan, a été d'abord introduit par les systèmes Music 3, Music 4 et Music 5 de Mathews, Csound de Vercoe et NetSound. Ce sont des langages de spécifications d'instruments de synthèse qui permettent de configurer des synthétiseurs afin de reproduire exactement une partition électronique. Le décodeur MPEG-4 SA (figure 12.7) est donc à la fois un *scheduler* et un synthétiseur. Dans ce modèle, le *bitstream* MPEG-4 SA est composé de deux parties :

– un *header* permettant la configuration et la paramétrisation du décodeur. En particulier, la définition des instruments est effectuée dans le langage SAOL (*Structured Audio Orchestra Language*) ;

– la partition à exécuter sous forme d'une séquence d'éléments temporels (access units). Cette partie est exprimée dans le langage SASL (*Structured Audio Score Language*).

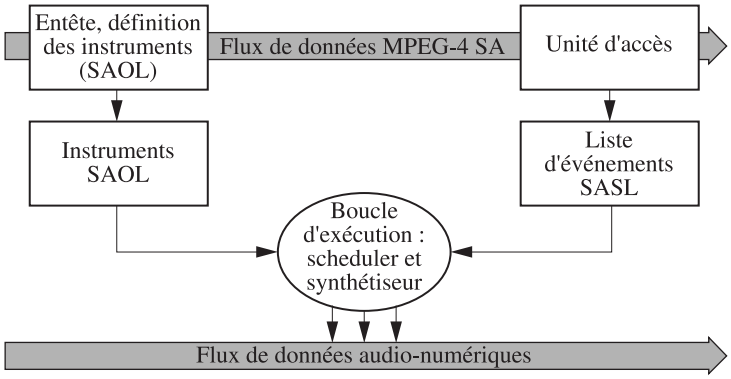


Figure 12.7. *Schéma général du décodeur MPEG-4 SA [SCH 99]*

Les langages SAOL et SASL permettent un spécification très précise des sons à synthétiser. Ils définissent deux échelles de temps : celles de l'échantillon (*a-rate*) et celle de la note (*k-rate*), dont les valeurs par défaut sont 32 kHZ et 100 Hz, respectivement. Cette approche permet une spécification unifiée de tous les paramètres de contrôle de la musique. A ce jour, deux implémentations de ces langages ont été rendues publiques [LAZ 99, SCH 99].

Dans le cas de simples sons samplés (sans synthèse), le codage SA peut paraître excessivement complexe. En conséquence, une version simplifiée a été développée avec le consortium MIDI : le SASBF (*Structured Audio Sample Bank Format*).

Ce format consiste à transmettre les tables d'onde, sous forme d'échantillons PCM échantillonnés à leur fréquence critique, et une liste d'événements midi tenant lieu d'événements SASL. De la sorte, on obtient un certain niveau de compatibilité avec le MIDI [SCH 99].

12.5.4. *MPEG-4 Text to speech*

Le troisième mode de codage de l'audio en MPEG-4 concerne la parole transcrite. Dans ce cas, ce n'est pas la parole enregistrée qui est transmise, mais les paramètres qui en permettent la synthèse : texte, phonèmes, rythme, prosodie, volume, hauteur, etc. MPEG-4 ne standardise que la syntaxe et le format d'encodage de ces paramètres, pas les techniques de synthèse vocale utilisées par le décodeur MPEG-4 qui dépendront donc des implémentations.

12.6. MPEG-7 et la description des sons et de la musique

Contrairement aux précédents standards, la norme MPEG-7 ne concerne pas directement les données audio et vidéo (*data*) mais les informations qui les accompagnent et qui les décrivent (*metadata*). C'est donc à juste titre que l'on parle d'une norme pour la *description* de documents multimédia. Lancés en 1998, la norme MPEG-7 a atteint le statut de standard international en 2002, et sera probablement complétée au cours du temps par des extensions et de nouveaux profils [HUN 99, NAC 99b].

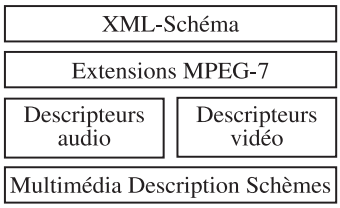


Figure 12.8. *Les quatre niveaux d'accès au standard MPEG-7*

Les objectifs de la norme MPEG-7 recouvrent toutes les dimensions de la *description* des documents multimédia, que ce soit à des fins de traitements, de recherche d'information ou d'interactivité. De plus, la norme n'institue pas pour l'instant de différence entre les descriptions extraites du contenu (*content-based*) et les descriptions en langue naturelle, bien que ces deux catégories présentent des caractéristiques très différentes.

En revanche, une distinction a été instituée dès les premières phases de préparation de la norme, entre plusieurs niveaux d'abstraction :

– descripteurs (D) correspondant à des éléments plus ou moins atomiques et indépendants du contexte ;

– schémas de description (DS) correspondant à des organisations spécifiques de descripteurs, adaptés à des contextes plus précis ;

– langage de définition des descriptions (DDL), permettant d'exprimer de nouveaux descripteurs et schémas de description tout en restant dans le cadre de la norme. Ce dernier aspect montre l'influence du monde du document électronique (normes SGML et XML) sur le groupe MPEG.

L'organisation générale de la norme MPEG-7 peut ainsi être décrite par la figure 12.8. Les quatre niveaux d'accès au standard MPEG-7, qui montre les différents niveaux d'accès possibles au standard. Dans cette structuration, chaque niveau utilise les niveaux inférieurs, mais reste indépendant des niveaux supérieurs. Le premier niveau contient la définition des schémas XML. A ceci s'ajoute dans un second niveau, quelques extensions permettant notamment de représenter des données numériques multidimensionnelles. A un troisième niveau, la norme définit les descripteurs audio et vidéo standard – dont la syntaxe et la sémantique ont été fixées par MPEG dans les parties 4 et 5 de la norme. Enfin, le quatrième niveau de la norme définit un grand nombre de constructions fort complexes et élaborées – les schémas de description multimédia, qui font l'objet des 800 pages de la partie 6 du standard. A partir de ces quatre grandes catégories de descriptions, la norme MPEG-7 permet d'unifier des applications très diverses, allant des moteurs de recherche aux hypertextes et au filtres d'informations [FOO 99, PFE 96].

12.6.1. *Métalangage de définition des descripteurs MPEG-7*

En suivant la voie tracée par les normes SGML et XML, MPEG a choisi d'inclure un métalangage dans la norme MPEG-7. Il s'agit d'un de XML-Schema, un autre standard élaboré sous l'égide du WWW Consortium (W3C) [MPEG 02]. Du point de la description des œuvres musicales, le standard pourrait ainsi renouveler l'actualité d'une norme déjà ancienne et sans doute un peu méconnue : SMDL (*Standard Music Description Language*) qui est à l'origine du standard HyTime. SMDL avait été initialement développé dans le cadre de SGML, pour susciter ensuite des extensions très importantes à ce standard, notamment dans le domaine de la structuration temporelle, point sur lequel SGML est naturellement faible (parce qu'adapté au texte écrit, séquentiel). Ces extensions ont donné lieu à un nouveau standard, HyTime, dont la complexité a jusqu'à présent empêché le développement. On peut donc spéculer sur l'avenir de MPEG-7 dans le cadre des langages bases sur XML-Schema. Il n'en reste pas moins acquis aujourd'hui que le

cadre formel des schémas XML est devenu – *via* MPEG-7 – un esperanto pour les diverses disciplines qui se partagent le terrain de l'indexation audiovisuelle.

12.6.2. *Descripteurs MPEG-7*

La recherche en analyse musicale par ordinateur peut tirer un bénéfice important de la possibilité de définir et de gérer des descripteurs normalisés, à différents niveaux de complexité. Ceci inclut les propriétés psycho-acoustiques prédites par les différents modèles de compression, mais aussi les estimations de grandeurs comme la fréquence fondamentale, la tonalité, ou les différents paramètres du timbre musical, et certaines interprétations sémantiques ou musicologiques provenant d'annotations manuelles : les sources sonores, les événements reconnaissables, les structures mélodiques et harmoniques.

Parmi les descripteurs audio standardisés par MPEG, on se contentera de citer ici les plus importants en renvoyant au chapitre 17 de [SMI 01] :

– les descripteurs de base sont *AudioWaveform* et *AudioPower*, qui indiquent simplement l'intervalle des amplitudes du signal audio et la moyenne de sa puissance instantanée. Un descripteur spécifique permet également de marquer un segment audio comme « silence » ;

– les descripteurs spectraux sont l'enveloppe spectrale (*AudioSpectrumEnvelope*) qui donne les puissances relatives des bandes spectrales, leur centroide (*AudioSpectrumCentroid*), leur variance (*AudioSpectrumSpread*), et le rapport pour chaque bande spectrale de la moyenne géométrique à la moyenne arithmétique des amplitudes (*AudioSpectrumFlatness*) ;

– les descripteurs harmoniques sont la fréquence fondamentale (*AudioFundamentalFrequency*) et l'harmonicité (*AudioHarmonicity*). Les méthodes utilisées pour déterminer la fréquence fondamentale sont laissées libres, mais doivent inclure une mesure de périodicité du signal pouvant servir à pondérer les résultats ;

– les descripteurs de timbre décrivent la structure temporelle d'un segment sonore sous la forme des descripteurs *LogAttackTime* et *TemporalCentroid*. Ils incluent en outre leurs propres définitions des descripteurs harmoniques selon une échelle linéaire, mieux adaptée à la caractérisation des timbres musicaux [PEE 00] ;

– les descripteurs de base spectrale permettent la réduction de données par les méthodes d'analyse en composantes principales (ACP) ou d'analyse en composantes indépendantes (ICA) [CAS 01a, CAS 01b].

Il est intéressant de constater que ces descripteurs sont souvent des innovations, des progrès par rapport à l'état de l'art qui prévalait dans les premières années de la standardisation MPEG-7.

12.6.3. *Schémas de description MPEG-7*

Les schémas de description multimédia introduisent une syntaxe pour la description structurale des signaux sonores :

- séparation en sources ou pistes logiques,
- segmentation temporelle selon plusieurs échelles de temps,
- composition hiérarchique des pistes et des segments.

Ces schémas ont également un volet sémantique. Ce niveau de description s'articule autour de deux notions délicates : celle d'objet sonore, qui concerne la description des sources sonores (instruments ou groupes d'instruments, voix, bruits et ambiances sonores), et celle d'événement sonore, qui décrit le comportement temporel d'un ou plusieurs objets sonores (note, phrase musicale).

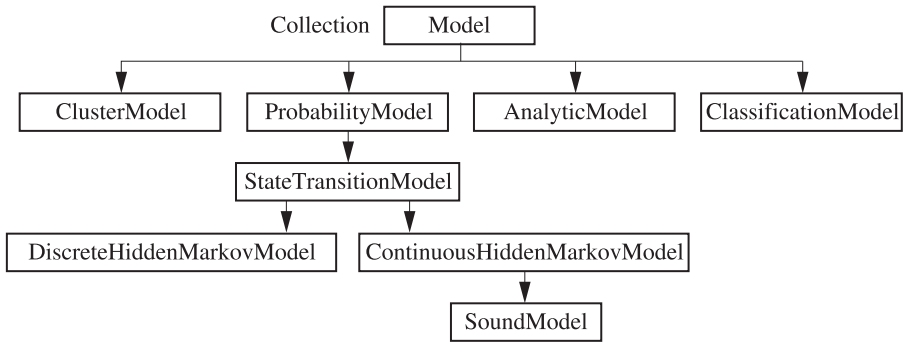


Figure 12.9. *Modèles pour la description sonore dans MPEG-7*

Dans le domaine de la sémantique, l'attitude générale de MPEG-7 consiste en une relative prudence. Plutôt que d'adhérer à une théorie ou une autre de la sémantique des objets multimédias, le standard définit la notion de modèles. Par exemple, la description peut emprunter les termes d'un thésaurus – qui sera alors référencé par MPEG-7 sous la forme d'un « modèle analytique ». Mais la même description peut également être donnée selon une représentation probabiliste – qui sera alors référencée par MPEG-7 sous la forme d'un « modèle probabiliste ». Nous avons représenté en figure 12.9 quelques-uns des modèles de descriptions définis par MPEG-7. En plus de ceux-ci, il convient de noter l'existence de schémas de description spécifiques à la représentation des transcriptions de la mélodie [KIM 00] et de la parole [CHA 00].

12.7. Conclusion

Les standards MPEG ont été développés dans le but de favoriser la transmission numérique des programmes audiovisuels, en assurant des économies d'échelles et en rationalisant les investissements des acteurs. Ce point a été acquis grâce à l'adoption de techniques innovantes ; celles-ci ont réussi à vaincre le problème d'obsolescence qui frappe habituellement ce genre d'initiatives. Il est à cet égard tout à fait remarquable de voir le groupe MPEG évoluer vers des directions nouvelles et de plus en plus innovantes et ambitieuses, comme le MPEG-4 et le MPEG-7, qui renforcent encore les liens entre la communauté scientifique des chercheurs en analyse musicale, et celle des industriels qui implémentent les standards de compression, de structuration et de description de la musique numérique.

12.8. Bibliographie

[BOS 97] BOSI M. *et al*., « ISO/IEC MPEG-2 Advanced Audio Coding », *J. Audio Engineering Society*, 45(10), 1997.

[BRA 91] BRANDENBOURG, K., « ASPEC: Adaptive Spectral Entropy Coding of High Quality Music Signals », *90th Convention of the Audio Engineering Society*, 1991.

[CAS 01a] CASEY M., « General Sound Similarity and Recognition: Musical Applications of MPEG-7 Audio », *Organized Sound*, 2001.

[CAS 01b] CASEY M., « MPEG-7 Sound-Recognition Tools », *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6), p. 737-747, 2001.

[CHA 00] CHARLESWORTH J., GARNER P., « Spoken content metadata and MPEG-7 », *ACM workshops on Multimedia*, Los Angeles, Californie, Etats-Unis, 2000.

[DEH 91] DEHERY P., LEVER M., « A MUSICAM Source Codec for Digital Audio Broadcasting and Storage », *Proceedings of Int. Conf. Acoustic, Speech, Signal Processing*, p. 3605-3608, 1991.

[FOO 99] FOOTE, J., « An Overview of Audio Information Retrieval », *Multimedia Systems*, 7(1), p. 2-10, 1999.

[GER 92] GERSHO A., GRAY R.M., *Vector quantization and signal compression*, Kluwer Academic Publishers, Norwell, 1992.

[HUN 99] HUNTER J., « MPEG-7 Behind the scenes », D-*Lib Magazine*, 5(9), 1999.

[KIM 00] KIM Y. *et al*., *Analysis of a contour-based representation for melody*, *International Symposium on Music Information Retrieval*, 2000.

[LAZ 99] LAZZARO J., WAWRZYNEK J., *The MPEG-4 Structured Audio Book*, 1999.

[LEV 96] LEVINE S., *Effects processing on audio subband data*, *Int. Computer Music Conference*, 1996.

[MAN 02] MANJUNATH B.S., SALEMBIER P., SIKORA T., *Introduction to MPEG-7*, 2002.

[MOO 89] MOORE B.C.J., *Introduction to the psychology of hearing*, Academic Press, New York, 1989.

[MPEG 92] MOTION-PICTURE-EXPERT-GROUP, *Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s*, 1992.

[MPEG 94] MOTION-PICTURE-EXPERT-GROUP, *Information technology – Generic coding of moving pictures and associated audio information,* (13818), 1994.

[MPEG 98] MOTION-PICTURE-EXPERT-GROUP, *Information technology – Coding of audio-visual objects,* (14496), 1998.

[MPEG 02] MOTION-PICTURE-EXPERT-GROUP, *Information technology – Multimedia content description interface (MPEG-7), Norme ISO/IEC,* (15938), 2002.

[NAC 99a] NACK F., LINDSAY A., « Everything You Wanted to Know about MPEG-7: Part 1 », *IEEE Multimedia*, 6(3), 1999.

[NAC 99b] NACK F., LINDSAY A., « Everything You Wanted to Know about MPEG-7: Part 2 », *IEEE Multimedia*, 6(4), 1999.

[PAI 00] PAINTER T., SPANIAS A., « Perceptual Coding of Digital Audio », *Proc. IEEE*, avril 2000.

[PAN 95] PAN D., « A tutorial on mpeg/audio compression standard », *IEEE Multimedia*, 260, 74, 1995.

[PEE 00] PEETERS G., MCADAMS S., HERRERA P., « Instrument Sound Description in the Context of MPEG-7 », *ICMC*, Berlin, 2000.

[PFE 96] PFEIFFER S., FISCHER S., EFFELSBERG W., « Automatic audio content analysis »., *ACM Multimedia*, Boston, Etats-Unis, 1996.

[SCH 99] SCHEIRER E.D., VNEN R., HUOPANIEMI J., « AudioBIFS: Describing Audio Scences with {MPEG}-4 Multimedia Standard », *IEEE Transactions on Multimedia*, 1(3), p. 237-250, 1999.

[SMI 01] SMITH J., SALEMBIER P., « MPEG-7 Multimedia Description », *Schemes IEEE Transactions on Circuits and Systems for Video Technology*, 11(6), p. 748-759, 2001.

[SWI 90] SWICKER E., FASTL H., *Psychoacoustics facts and models*, Springer Verlag, Berlin, 1990.

[WAT 99] WATKINSON J., *MPEG-2*, 1999.

[WOL 99] WOLD E. *et al*., *Classification, Search and Retrieval of Audio*, *Handbook of Multimedia Computing*, p. 207-225, 1999.

Chapitre 13

Les normes MIDI et MIDIFiles

13.1. Les normes MIDI et MIDIFiles

13.1.1. *Historique*

Dans un studio audionumérique, sont utilisées classiquement des machines de production ou de traitement du son : synthétiseur, échantillonneur, multi-effets et table de mixage, ainsi que des appareils de commande ou de contrôle : clavier maître, boite à rythme, ordinateur. On peut alors distinguer deux grandes familles de flux de données :

– des données audio avec par exemple une chaîne du type : prise de son, enregistrement et stockage sur bancs de montage numérique, traitement (effets, spatialisation), mixage et enfin diffusion ;

– des données de contrôle avec par exemple une chaîne du type : clavier maître, ordinateur, échantillonneurs et expandeurs.

Les premiers synthétiseurs communiquaient en échangeant des signaux analogiques pour transmettre la hauteur des notes jouées aux générateurs sonores, ou des signaux d'horloges pour se synchroniser. Dans les années 1980, des instruments de musique digitaux contenant des microprocesseurs et des mémoires sont apparus. La norme MIDI (*Musical Instrument Digital Interface*) [MIDI] a été définie alors dans un souci de normalisation des *protocoles de contrôle* utilisés par ces différentes machines. Elle a été adoptée très rapidement par l'ensemble des constructeurs à partir de 1983 et a permis une rapide standardisation des matériels.

Chapitre rédigé par Stéphane LETZ.

C'est aussi devenu, dans sa déclinaison « fichier » (la norme MIDIFile), un support de la musique nécessitant une très faible bande passante, utilisé par exemple dans les jeux ou applications multimédia. Un fichier MIDIFile prend peu de place, son contenu musical peut facilement être transformé (transposition, changement de tempo, etc.), alors que ces opérations sont nettement plus difficiles à opérer sur des fichiers audio préenregistrés.

Enfin, la réussite de ce protocole a amené par la suite à en définir des extensions : soit sous la forme de sous-normes précisant certains points particuliers comme les protocoles de synchronisation (*MIDI Time Code, MIDI Machine Control*), de gestion des sons (*General MIDI*) ou utilisées dans des secteurs annexes : contrôle de périphériques lumières ou dispositifs multimédia.

13.1.2. *Présentation de la norme MIDI*

La norme MIDI est un protocole pour la transmission d'informations entre plusieurs machines, créée initialement pour les instruments de musique électriques. Apparue officiellement en 1983, cette norme réglemente la communication entre différentes catégories d'appareils : synthétiseurs, échantillonneurs, boîtes à rythmes, ordinateurs, magnétophones, etc. Elle est composée de trois parties : un format de connecteur, un protocole de commande et enfin un format de fichiers de distribution de séquences musicales : le format MIDIFile.

13.1.2.1. *Connecteurs*

La norme MIDI est en premier lieu une interface standardisée pour connecter physiquement des appareils entre eux. Chaque instrument équipé de prises MIDI peut être un émetteur et un récepteur. Il existe trois types de connecteurs MIDI : entrée (*MIDI In*), sortie (*MIDI Out*) et redirection (*MIDI Thru*) (figure 13.1). Deux câbles doivent être utilisés pour réaliser une communication bidirectionnelle. Chaque câble MIDI peut transporter seize canaux logiques en utilisant, pour la majorité, des messages ayant une valeur de canal codée sur 4 bits. Un module de son pourra alors jouer de manière indépendante seize instruments différents.

13.1.2.2. *Protocole*

A la base, le protocole MIDI permet d'encoder sous la forme de messages successifs, un *geste musical* effectué sur un instrument MIDI. Typiquement, un clavier envoie des messages à chaque fois qu'une note est appuyée, avec le numéro de la note et la vitesse d'enfoncement de la touche. Des messages *contrôleurs* sont utilisés pour décrire des paramètres à évolution continue comme le volume ou le glissando. Le protocole est suffisamment riche pour décrire tout ce qui est de l'ordre du contrôle : geste musical, variation de paramètres, état des machines, etc.

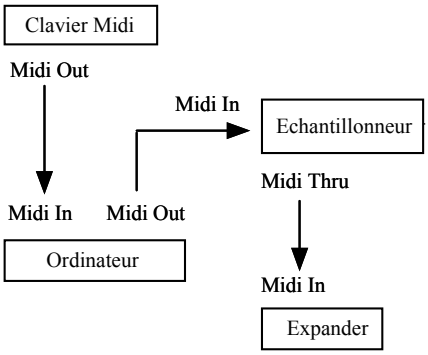


Figure 13.1. *Connections entre instruments MIDI*

MIDI est un protocole de commande série asynchrone à 31,25 kbaud. Le format de transmission est constitué de paquets de 10 bits : un bit de début, huit bits de données et un bit de fin. Le protocole de communication MIDI est réalisé à l'aide de messages multi-octets de deux types :

– octet d'état (*status byte*) dont le bit 7 est à 1, de la forme *1nnnnnnn* en binaire. Lorsqu'un octet de ce type est reçu, le récepteur reste dans l'état courant jusqu'à la réception d'un nouvel octet d'état. Ainsi, il est possible d'économiser de la bande passante en omettant l'octet d'état si plusieurs messages du même type se suivent ;

– octet de donnée dont le bit 7 est à 0, de la forme *0ddddddd* en binaire. Les messages comportent 0, 1, 2 octets de données pour les « petits » messages et un nombre variable d'octets pour les messages systèmes exclusifs.

Les messages sont divisés en deux catégories principales :

– les *messages canaux* qui sont adressés spécialement à l'un des seize canaux MIDI à l'aide des 4 bits de poids faible de l'octet d'état. Ils seront reçus par les instruments configurés pour recevoir sur ce canal. On distingue :

- les *messages voies* constitués de 7 types différents qui servent à contrôler les paramètres musicaux. Par exemple, le message *NoteOn* dont le codage en hexadécimal est le suivant : 90 (à 9F les canaux étant codés sur les 4 bits de poids faible), suivi d'un numéro de note sur 7 bits de 0 à 127, et d'une vélocité de 0 à 127 ;

- les *messages modes* qui définissent les modes de réaction de l'instrument aux messages voix (omni, poly, mono). Par exemple, le message *AllNoteOff* pour arrêter toutes les notes en cours de jeu ;

– les *messages systèmes* qui ne contiennent pas d'information de canal et sont donc destinés à tous les appareils connectés. On distingue :

- les *messages communs* destinés à tous les appareils. On trouve par exemple dans cette catégorie les messages de synchronisation (*Quarter Frame* ou de changement d'accord, *Tune Request* codé F6 en hexadécimal) ;

- les *messages temps-réel*, leur caractéristique principale est de pouvoir être insérés n'importe où dans un flux MIDI. Ils sont utilisés pour la synchronisation (par exemple le message *Start* codé F8 en hexadécimal) ;

- les messages *systèmes exclusifs*, utilisés pour transmettre des donnés de longueur variable, par exemple l'état des mémoires des appareils. Ils comportent un en-tête qui spécifie leur type, un identificateur constructeur, etc., suivi des données effectives et d'un octet de fin de message.

13.1.3. *Extensions et sous-normes*

On donne ici quelques exemples de sous-normes couramment utilisées.

13.1.3.1. *SDS, Sample Dump Standard*

Beaucoup d'instruments utilisent des échantillons comme source sonore. Un sous-protocole spécifique a été défini dans la norme MIDI de façon à permettre l'échange d'échantillons entre ce type d'appareils sous la forme de messages MIDI standard. Ce protocole utilise des messages systèmes exclusifs particuliers, il est implémenté dans de nombreux échantillonneurs.

13.1.3.2. *MTC*

Le protocole *MIDI Time Code* (MTC) est un sous-protocole de la norme MIDI qui permet de synchroniser différents appareils. C'est une alternative au protocole utilisant les messages *clocks* MIDI et *Song Position Pointers*, en réalité le format SMPTE transformé pour être transmis sous la forme de messages MIDI. Une date SMPTE est un temps absolu décrit sous la forme heures, minutes, secondes, images (une subdivision de la seconde). Au contraire, le format clocks MIDI et *Song Pos* décrit le temps en mesures à partir du début de la séquence avec une notion de tempo. Le protocole SMPTE est constitué de messages particuliers (*Quarter Frame*) ainsi que de plusieurs systèmes exclusifs spécifiques.

13.1.3.3. *General MIDI*

La norme *General MIDI* a été définie afin de garantir qu'un MIDIFile utilisant des messages de changement de programme, soit toujours joué de la même manière, c'est-à-dire avec les mêmes instruments, quelque soit le module de sons utilisé.

La norme General MIDI établit une correspondance entre des numéros de programme et des familles de sons d'instruments. Par exemple, le numéro de programme 1 correspondra toujours à un son de « grand piano acoustique ». Les

types d'instruments sont regroupés en seize familles de huit sons. Par exemple, la famille des cordes comprend le violon (numéro 41), l'alto (42), le violoncelle (43), la contrebasse (44), etc. Un module General MIDI contient aussi un ensemble de sons de batterie/percussions où la correspondance entre les numéros de note et les différents sons est précisément spécifiée.

13.1.3.4. *MMC*

La norme *MIDI Machine Control* (MMC) a été définie spécifiquement pour contrôler à distance des systèmes d'enregistrement audio *direct to disk* ou d'autres types d'appareils utilisés en enregistrement ou diffusion, en utilisant des messages MIDI systèmes exclusifs. La norme a été étendue de façon à contrôler d'autres types d'appareils comme des lumières et des machines d'effets (*MIDI Show Control*).

13.1.4. *La norme MIDIFile*

La norme MIDI décrit des messages reçus et interprétés en temps réel par les appareils récepteurs. Lorsque ces données sont stockées dans un fichier, il est nécessaire de leur ajouter une information temporelle sous la forme d'une date.

La norme *MIDIFile* est un format de fichier pour le stockage et l'échange de données MIDI datées utilisés par les séquenceurs matériels ou logiciels. Ce format permet de stocker les informations MIDI standard avec une date pour chaque message, qui code la différence avec la date du message précédent (la date est codée sous un format à *longueur variable* qui permet d'utiliser le nombre minimum d'octets nécessaires). Le format autorise le stockage d'informations supplémentaires absentes de la norme MIDI comme des valeurs de tempo, de résolution temporelle, de signature temporelle, de clef, de nom des pistes, etc.

Il est suffisamment générique pour permettre à toutes les applications musicales de créer et lire des fichiers sans perdre les informations les plus importantes, et suffisamment souple pour qu'une application particulière puisse stocker des informations propriétaires que d'autres applications pourront ignorer lors du chargement.

13.1.4.1. *Formats*

Trois formats ont été définis :

– un fichier au format 0 contient un en-tête suivi d'une piste unique. C'est le format le plus commun et qui est généralement utilisé par tous les séquenceurs ;

– les fichiers au format 1 ou 2 contiennent un en-tête suivi d'une ou plusieurs pistes. Les programmes qui définissent plusieurs pistes utilisent généralement le format 1, le format 2 est réservé aux programmes qui manipulent des « pattern ».

13.1.4.2. *Données*

Un fichier MIDIFile est formé d'une suite de blocks (*chunks*). Un block contient un identificateur de quatre caractères, suivi de la taille des données contenues dans le block (codé sur 32 bits), suivi des données elles-mêmes.

La norme MIDIFile définit deux types de blocks : block *d'en-tête* et block de *piste*. Le block d'en-tête contient les informations minimales nécessaires pour interpréter l'ensemble du fichier : format du fichier, nombre des pistes, résolution et format temporel (les dates sont exprimées soit en temps musical avec tempo, soit en temps absolu). Le block de piste contient la suite des données MIDI (ou des messages spécifiques MIDIFile) qui peuvent appartenir aux seize canaux MIDI, précédées d'une date.

Les fichiers multipistes contiennent un block d'en-tête suivi de plusieurs blocks de pistes. Les informations de tempo et de signature temporelle sont contenues dans une piste particulière appelée la *table de tempo* (*Tempo Map*), toujours stockée dans la piste 0. Dans le cas d'un MIDIFile au format 0, les informations de tempo et de signature temporelle sont mélangées avec les autres types d'événements sur une piste unique.

13.1.5. *Parsing d'un flot MIDI*

La spécification MIDI définit un protocole spécial utilisé pour la transmission de messages voix et de message mode appelé le *protocole avec état courant* qui permet d'économiser de la bande passante. Il est optionnel lors de la transmission mais tout récepteur doit l'implémenter.

Implémentation d'un parseur MIDI

Le récepteur d'un flot MIDI doit être capable de gérer le protocole avec état courant. Pour cela, il conserve la valeur de l'octet d'état courant. Le récepteur doit, de plus, connaître le nombre d'octet de données qui sont attendues pour chaque type de message.

Les messages temps réels, utilisés pour la synchronisation et constitués d'un seul octet, peuvent être intercalés n'importe où dans le flot MIDI. Le parseur (analyseur syntaxique) doit donc déterminer pour chaque octet reçu :

– si celui-ci est un message temps réel, il est traité immédiatement ;

– si c'est un octet faisant partie d'un message MIDI de plusieurs octets en cours de réception.

On implémente classiquement un parseur MIDI à l'aide d'une fonction qui analyse chaque octet et applique le traitement correspondant en fonction de sa valeur. Nous présentons ici un parseur optimisé, utilisé dans le système MidiShare [ORL 89]. C'est une machine à état qui utilise une *continuation* sous la forme d'une fonction à appeler à chaque octet reçu, en évitant ainsi l'analyse exhaustive de tous les états possibles.

La structure de donnée parseur est la suivante :

```
typedef struct StreamFifo{
   ParseMethodPtr    parse ; // continuation : fonction à
exécuter
   ParseMethodPtr * rcv ;   // tables des méthodes
   // champs supplémentaires pour stocker les données en
cours
} StreamFifo ;
```

Elle contient :

– une *continuation* courante, c'est-à-dire une fonction à appeler lors de la réception du prochain octet ;

– une *table de méthode* associant une méthode pour chaque type d'octet d'état ;

– des champs supplémentaires pour stocker les données intermédiaires reçues. Ils ne sont pas explicités ici.

La fonction `MidiParseInit` initialise la structure `StreamFifo` :

```
void MidiParseInit (StreamFifo * f, ParseMethodTbl rcv)
{
 f->parse     = rcvStatus ;
 f->rcv       = rcv ;  // table de méthodes
 // initialise les champs supplémentaires
    // pour stocker les données en cours
}
```

La table des méthodes contient les fonctions suivantes :

– `rcvChan2` pour la réception des messages canaux à 2 octets,

– `rcvChan1` pour la réception des messages canaux à 1 octet,

– `rcvCommon2` pour la réception des messages communs à 2 octets,

– `rcvCommon1` pour la réception des messages canaux à 1 octet,

– `rcvCommon0` pour la réception des messages canaux sans octet de donnée,

– `rcSysExBeg` pour la réception des messages Système exclusif,

– `rcQFrameg` pour la réception des messages *Quarter Frame*.

Notons que certaines d'entre elles font appel à des méthodes auxiliaires pour traiter les octets qui suivent l'octet d'état reçu :

- `rcvDataM` pour la réception du 1° octet de donnée d'un message à 2 données,
- `rcvDataD` pour la réception du 2° octet de donnée d'un message à 2 données,
- `rcvDataU` pour la réception de l'octet de donnée d'un message à 1 donnée,
- `rcvDataQ` pour la réception de l'octet de donnée d'un message *Quarter Frame*.

Par exemple, `rcvChan2` utilise les méthodes `rcvDataM` et `rcvDataD` pour traiter les deux octets de donnée. Ainsi, le message canal *KeyOn* suivi des données de hauteur et vélocité sera traité par la fonction `rcvChan2`.

La fonction `rcvStatus` est appelée à chaque octet reçu et exécute la continuation :

```
static bool rcvStatus (StreamFifo* f, char c)
{
 // type(c,f) retourne le type associé à l'octet MIDI
 return (c < 0) ? (*f->rcv[type(c, f)])(f, c) : false ;
}

static bool rcvStore (StreamFifo* f)
{
 // alloue une nouvelle cellule pour l'événement en cours
de réception
 return true ;
}
```

Enfin, les méthodes de réception des différents types de messages sont décrites ici :

```
// rcvChan2 : réception d'un message canal à 2 octets
static bool rcvChan2 (StreamFifo* f, char c)
{
 // conserver le type et le canal
 f->parse = rcvDataM ;
 return false ;
}

// réception de la 1° donnée
static bool rcvDataM (StreamFifo* f, char c)
{
 if(c < 0) return rcvStatus(f, c) ;
 // conserver la date courante et la donnée
 f->parse = rcvDataD ;
 return false ;
}
```

```
// réception de la 2° donnée
static bool rcvDataD (StreamFifo* f, char c)
{
 if( c < 0) return rcvStatus(f, c) ;
 // conserver la donnée
 f->parse = rcvDataM ;
 return rcvStore(f) ;
}

// rcvChan1 : réception d'un message canal à 1 octet
static bool rcvChan1 (StreamFifo* f, char c)
{
 // conserver le type et le canal
 f->parse = rcvDataU ;
 return false ;
}

// réception de l'octet de donnée
static bool rcvDataU (StreamFifo* f, char c)
{
 if(c < 0) return rcvStatus(f, c) ;
 // conserver la date courante et la donnée
 return rcvStore(f) ;
}

// rcvCommon2 : réception d'un message commun à 2 octets
static bool rcvCommon2 (StreamFifo* f, char c)
{
 // conserver le type et le canal
 f->parse = rcvDataM ;
 return false ;
}

// rcvCommon1 : reception d'un message commun à 1 octet
static bool rcvCommon1 (StreamFifo* f, char c)
{
 // conserver le type et le canal
 f->parse = rcvDataU ;
 return false ;
}

// rcvCommon0 : réception d'un message commun sans octet
de donnée
static bool rcvCommon0 (StreamFifo* f, char c)
{
 // créer l'événement
 return true ;
```

```
}

// rcvQFrame : réception d'un message quarter frame
static bool rcvQFrame (StreamFifo* f, char c)
{
 // conserver le type et le canal
 f->parse = rcvDataQ ;
 return false ;
}

// réception des données d'un message quarter frame
static bool rcvDataQ (StreamFifo* f, char c)
{
 if(c < 0) return rcvStatus(f, c) ;
 // conserver la date courante et la donnée
 return rcvStore(f) ;
}

// rcvSysExBeg : réception d'un System Exclusif
static bool rcvSysExBeg (StreamFifo* f, char c)
{
 // créer l'événement System Exclusif
 f->parse = rcvSysExNext ;
 return false ;
}

// réception des données d'un System Exclusif
static bool Ptr rcvSysExNext (StreamFifo*f, char c)
{
 if(c < 0) return rcvSysExEnd(f, c) ;
 //  ajouter  la  donnée  à  l'événement  en  cours  de
construction
 return false ;
}

static bool rcvSysExEnd (StreamFifo*f, char c)
{
 if((unsigned  char)c  >=  (unsigned  char)0xf8)  //  MIDI
clock
        return rcvStatus (f, c) ;
 if(c != (char)EndSysX)
        return rcvStatus (f, c) ;

 // terminer la création de l'événement System Exclusif
 f->parse= rcvStatus ;
 return true ;
}
```

Notons que cette implémentation est donnée à titre indicatif. Elle n'explicite pas entièrement comment les données intermédiaires sont mémorisées et les messages MIDI complets sont construits.

13.2. Représentation du temps et synchronisation

13.2.1. *Représentation du temps*

Dans les applications musicales, plusieurs représentations du temps cohabitent, chacune donnant une lecture du contenu sonore plus ou moins adaptée à une utilisation particulière. Le temps est habituellement représenté de deux manières :

– temps absolu, avec par exemple le format SMPTE, qui est une représentation en heures, minutes, secondes, et images (une subdivision de la seconde) ;

– temps musical qui s'exprime en *ticks* avec une valeur de tempo. Cette valeur sera le plus souvent représentée sous un format mesure, temps, ticks directement lié à la représentation solfégique connue du musicien.

Si par exemple on travaille sur un enregistrement sonore, une représentation en temps absolu sera la plus adaptée. En revanche, un éditeur de partition en notation musicale traditionnelle donnera la préférence à la représentation en temps musical.

Lorsque l'utilisateur travaille avec les deux représentations en temps musical et en temps absolu, celles-ci doivent toujours rester cohérentes. Il pourra par exemple spécifier la position courante en utilisant indifféremment l'une ou l'autre de ces représentations et le système s'occupe de réaliser les conversions nécessaires pour passer à l'autre représentation. Dans un séquenceur, on utilise classiquement une valeur interne en nombre de ticks, dont il sera possible de dériver :

– la valeur solfégique en mesures, temps, division,

– la valeur en temps absolu.

13.2.1.1. *Conversion temps interne vers temps musical*

La conversion entre la date interne en ticks et sa représentation en mesure, temps, division (et inversement) est faite en utilisant :

– la notion de *signature temporelle* (contenu dans le message MIDIFile *Time Sign*), c'est-à-dire un nombre d'une unité donnée par mesure (par exemple 3/4 signifie trois noires (1/4) par mesures, 12/8 signifie douze croches (1/8) par mesure). La valeur de la ronde est l'unité et les valeurs plus petites (blanches, noires, croches) sont exprimées sous la forme de fractions ;

– la notion de *résolution temporelle* (contenue dans l'en-tête du MIDIFile), c'est-à-dire le nombre de ticks pour une noire (*Ticks Per Quarter* ou TPQ). La

résolution temporelle est la finesse avec laquelle un séquenceur peut différencier des événements.

Par exemple, si `TPQ = 480`, c'est-à-dire 480 ticks par noire et la signature temporelle 3/4, c'est-à-dire 3 noires par mesure, alors on a pour une date *5 mesures, 2 temps*, *100 ticks* convertie en ticks :

```
 date en ticks = 5 * mesure exprimée en ticks + 2 * temps
exprimé en ticks + 100
        = 5 * (480 * 3) + 2 * 480 + 100
        = 8260 ticks
```

13.2.1.2. *Conversion temps interne vers temps absolu*

La conversion entre une date interne en ticks et sa représentation en temps absolu *hrs:min:sec:milli* est faite en utilisant :

– la notion de *tempo* qui s'exprime pour le musicien en « battues par minute » (beat-per-minute ou BPM) et généralement en microsecondes par noire dans les représentations internes, en particulier dans les MIDIFiles ;

– la notion de *résolution temporelle,* c'est-à-dire le nombre de ticks par noire.

A partir de la valeur du tempo et de la résolution temporelle en ticks par noire, il est possible de calculer la durée d'un tick en microsecondes.

Par exemple, si `TPQ = 480`, c'est-à-dire 480 ticks par noire avec un tempo interne de 1 000 000 microsecondes par noire (correspondant à 60 BPM), on a :

```
durée d'un ticks en microsecond   = 1000000/480
                                        = 2083,333
```

La date exprimée en ticks est ensuite convertie en microsecondes. Ainsi pour une date de 2 000 ticks, on aura :

```
date en microsecondes       = date en tick * durée d'un
tick
                            = 2083,333 * 2000
                            = 4166666,67
```

Cette valeur est enfin exprimée dans un format plus lisible, par exemple :

```
 4166666,67 = 0:0:4:167 au format hrs:min:sec:milli
```

Notons que ces exemples sont donnés à titre indicatif, les calculs internes devant être faits avec des représentations sans arrondis pour garder une précision parfaite à chaque étape.

13.2.2. *Synchronisation*

On peut voir deux aspects essentiels de la synchronisation :

– démarrer/arrêter un ensemble de machine en même temps,

– conserver une vitesse de lecture identique entre les appareils.

Si des machines doivent rester synchronisées, habituellement l'une se comporte comme « maître » et les autres comme « esclaves », asservies à la vitesse du maître ainsi qu'à ses ordres de démarrage et d'arrêt.

En interne, un séquenceur synchronisable utilise classiquement des dates exprimées en temps musical (ticks). Ces valeurs seront converties en temps absolu à l'aide d'un « synchroniseur » qui réalise la conversion du temps en ticks en temps absolu, à la volée, lors du rendu temporel des événements MIDI.

Nous allons distinguer ici essentiellement trois formes de synchronisation.

13.2.2.1. *Synchronisation interne*

C'est une forme un peu particulière au sens où la vitesse du séquenceur sera asservie à la lecture d'une « table de tempo » interne : c'est une séquence qui contient des événements tempo et signature temporelle datés, c'est-à-dire une fonction de déformation du temps qui permet à chaque instant de connaître la relation temps musical/temps absolu.

En mode synchronisation interne, le séquenceur asservit son tempo sur la table de tempo en changeant la valeur courante au passage de chaque événement tempo. De même, les changements de signature temporelle seront interprétés pour le calcul de la date en temps musical exprimés en mesures, temps et ticks. La connaissance de cette « table de tempo » permet aussi de connaître à chaque instant les dates exprimées en temps absolu et temps musical en utilisant les méthodes de calculs décrites précédemment.

13.2.2.2. *Synchronisation MIDI Sync*

C'est une forme de synchronisation qui utilise les messages MIDI *Start*, *Stop*, *Continue*, *SongPos* et *Clock* et qui fonctionne en temps musical : le maître produit vingt-quatre messages clocks par noire à une vitesse qui est fonction du tempo. Le séquenceur esclave avance sa date interne de 1/24 de noire à la réception de chaque clock et joue tous les événements correspondants, en interpolant si nécessaire, le tempo entre deux messages clocks consécutifs.

Les messages *Start*, *Stop*, *Continue* permettent respectivement de démarrer le séquenceur esclave à partir de la date 0, de l'arrêter, de le démarrer à partir de la

position courante. Enfin, le message *SongPos* est utilisé pour positionner le séquenceur esclave à une valeur donnée, avec une résolution de six *SongPos* par noire.

13.2.2.3. *Synchronisation SMPTE*

Le format SMPTE (*Society of Motion Picture and Television Engineers*) est un standard pour la synchronisation des données audio, film et vidéo. C'est un signal audio qui transporte des données digitales décrivant les dates en heures, minutes, secondes, images et sous-images qui sont des sous-divisions de la seconde. Le dispositif maître lit le code SMPTE qui est interprété par la machine esclave. Pour les appareils MIDI, une conversion intermédiaire est souvent utilisée : la date SMPTE est convertie en date *MIDI Time Code* (MTC), exprimée sous la forme de messages MIDI *Quarter Frame*. Une date complète est constituée de huit messages envoyés toutes les deux images, et le dispositif esclave a la possibilité de se synchroniser à chaque message *Quarter Frame* intermédiaire.

13.3. Limitations et évolution

Apparue dès le début des années 1980, la norme MIDI et ses différentes extensions ont été massivement adoptées par les constructeurs de matériels et logiciels, et ont fait preuve d'une remarquable stabilité au cours du temps. Malgré ce succès, et avec l'évolution continue des matériels et des besoins, certaines limitations sont apparues :

– bande passante limitée d'un canal de transmission MIDI pour des besoins de contrôles nécessitant des flux de données importants,

– définition de la norme autour d'un accès gestuel type clavier, mal adapté pour d'autres instruments,

– résolution parfois trop faible de certains types de messages,

– une architecture de connexion parfois lourde à mettre en œuvre,

– en tant que format de fichier, le format MIDIFile, bien que très utilisé dans la transmission de *performances musicales*, n'est pas bien adapté à la description de la *notation musicale* sous forme de partitions.

Des tentatives d'évolution visant à supprimer ces limitations sont apparues dans les années 1990. On peut citer ZIPI [MCM 94], un protocole de contrôle plus complet basé sur une architecture réseau, qui étend la résolution et le bande passante, mais qui n'a pas été réellement adopté.

Parmi les différentes évolutions actuelles, certaines gardent le même protocole (ou des extensions mineures) sur des supports à plus grande bande passante. On peut citer :

– mLAN défini par Yamaha, un protocole pour la transmission de données multimédia (audio, vidéo, MIDI, etc.) utilisé sur des connections de type *FireWire* ;

– MWPP [LAZ 03] une extension du protocole RTP [SCH 96] pour le transport de messages MIDI en temps réel sur des réseaux et notamment Internet.

D'autres évolutions s'attaquent plus fondamentalement aux autres limitations de la norme MIDI comme par exemple les problèmes d'adressage et de résolution. On peut citer : *Open Sound Control* OSC [WRI 97], un protocole de contrôle indépendant du support de transmission, avec un modèle d'adressage sophistiqué et des données datées définies avec une résolution étendue (32 ou 64 bits).

Enfin, de nouveaux formats destinés à remédier aux limitations du format MIDIFile pour la représentation de séquences musicales sont apparus. On peut citer : MusicXML [MUS 01], un format XML d'échange de séquences qui prend en compte le contenu musical ainsi que les informations relevant de la notation musicale, utilisé comme format d échange pour les séquenceurs et les logiciels d'édition ou d'analyse de partition.

Ces nouvelles technologies sont prometteuses mais encore assez peu adoptées. La famille des normes MIDI, par sa grande diffusion et sa stabilité, reste incontournable autant pour les matériels que les logiciels.

13.4. Bibliographie

[LAZ 03] LAZZARO J., WAWRZYNEK J., RTP Payload Format for MIDI, Internet-Draft, IETF, http://www.ietf.org/internet-drafts/draft-ietf-avt-mwpp-midi-rtp-07.txt, 2003.

[MES 97] MESSICK P., *Maximum MIDI in C++*, Softbound, New York, 1997.

[MUS 01] MusicXML Document Type Definition: Recordare LLC, www.recordare.com, 2001.

[WRI 97] WRIGHT M., FREED A., « Open SoundControl: A New Protocol for Communicating with Sound Synthesizers », *Proceedings of ICMC*, p. 101-104, 1997.

[ORL 89] ORLAREY Y., LEQUAY H., « MidiShare: a Real Time multi-tasks software module for Midi applications », *Proceedings of the ICMC*, San Francisco, 1989.

[MIDI] MIDI Manufacturers Association, The complete MIDI 1.0 Detailed Specification, Los Angeles.

[SCH 96] SCHULZRINNE H., CASNER S., FREDERICK R., JACOBSON V., RTP: A Transport Protocol for Real-Time Applications, Audio-Video Transport Working Group, RFC 1889, janvier 1996.

[MCM 94] MCMILLEN K, « ZIPI : Origins and Motivations », *Computer Music Journal*, vol. 18, n° 4, 1994.

Annexe

A.1. L'algorithme MDFL (version améliorée)

Une séquence mélodique est convertie en plusieurs profils d'intervalle paramétriques indépendants P_k pour les paramètres suivants : *hauteur* (intervalles de hauteurs), *IEII* (intervalles entre instants initiaux) et *silences* (intervalles entre la fin d'un événement et le début du suivant).

Les intervalles de hauteur peuvent être mesurés en demi-tons et les intervalles temporels (pour les IEII et les silences) en millisecondes ou quantifiés numériquement. Il faut fixer des limites maximales, comme la durée de la ronde pour les IEII et les silences, et l'octave pour les intervalles de hauteur ; les intervalles qui dépassent le seuil sont tronqués à la valeur maximale.

Un profil paramétrique P_k est représenté comme une séquence des intervalles n de taille x_i :

$$P_k = [x_1, x_2 \ldots x_n]$$

où $k \in$ {hauteur, IEII, silence }, $x_i \geq 0$ et i $\in \{1, 2 \ldots n\}$.

Le degré de changement r entre deux valeurs d'intervalles successifs x_i et x_{i+1} est :

$$r_{i,i+1} = \frac{|x_i - x_{i+1}|}{x_i + x_{i+1}} \text{ ssi } x_i + x_{i+1} \neq 0 \text{ et } x_i, x_{i+1} \geq 0$$

$$r_{i,i+1} = 0 \text{ ssi } x_i = x_{i+1} = 0$$

REMARQUE.– Il peut être utile d'ajouter une petite valeur à la taille de tous les intervalles, par exemple un demi-ton aux intervalles de hauteur, pour éviter les irrégularités apportées par les intervalles de taille nulle.

Le score de la frontière S_i pour l'intervalle x_i dépend du degré de différence par rapport à l'intervalle précédent et à l'intervalle suivant, d'après la fonction :

$$s_i = x_i \cdot \left(r_{i-1,i} + r_{i,i+1} \right)$$

Pour chaque paramètre k, on calcule la séquence $S_k = [s_1, s_2 \ldots s_n]$ et on la normalise dans l'intervalle de valeurs [0,1].

Le PFG d'une mélodie est la moyenne pondérée des séquences de force S_k (dans ce chapitre, nous utilisons les poids : $w_{hauteur} = 0{,}25$, $w_{IEII} = 0{,}50$ et $w_{silence} = 0{,}25$). Les frontières locales correspondent aux maxima locaux de cette séquence de scores.

A.2. L'algorithme Unscramble

Soit T un ensemble d'entités et P l'union de tous les ensembles des propriétés pertinentes pour la description de chaque entité. Si $d(x, y)$ est la distance entre deux entités x et y, et h est un seuil de distance, on définit la similitude $sh(x, y)$ comme suit :

$$\begin{aligned} &sh(x, y) = 1 \text{ ssi } d(x, y) \leq h \text{ (objets similaires)} \\ &sh(x, y) = 0 \text{ ssi } d(x, y) > h \text{ (objets non similaires)} \end{aligned} \qquad [A.1]$$

Autrement dit, deux entités sont *similaires* si la distance entre eux est inférieure à un seuil donné, et *non similaires* si leur distance est plus grande que ce seuil.

Pour un ensemble d'entités T, un ensemble de propriétés P et un seuil de distance h, une catégorie C_k est définie comme un ensemble maximal :

$$C_k = \{x_1, x_2 \ldots x_n / x_i \in T\} \text{ tel que } \forall \iota, j \hat{I} i,j \in \{1,2 \ldots n\}, s_h(x_i,x_j) = 1 \qquad [A.2]$$

Autrement dit, une catégorie C_k est un ensemble maximal d'entités similaires pour un seuil h.

Ces définitions de similitude et de catégorie peuvent être réexprimées dans la terminologie de la théorie des graphes de la façon suivante : les objets sont représentés par des nœuds dans un graphe non dirigé, la similitude entre objets similaires est représentée par les arcs, et les catégories sont les cliques maximales (une clique maximale est un sous-graphe maximal entièrement connecté).

Les données de l'algorithme Unscramble sont un ensemble d'objets N décrits chacun par un vecteur de propriétés à m dimensions (par exemple, l'objet x : *[*$p1$*,* $p2$ *...* p_m*]* et l'objet y : *[*$q1$, $q2$ *...* q_m*])*. Chaque propriété a un poids initial w_p=1.

Les propriétés représentent des traits binaires correspondants à une paire attribut-valeur. La distance entre deux objets est donnée par la fonction suivante (basée sur une distance de Hamming) :

$$d(x,y) = \sum_{i=1}^{m} w_{p_i} \cdot w_{q_i} \cdot \delta(p_i, q_i) \qquad [A.3]$$

où $\delta(p_i,q_i) = 0$ si $p_i = q_i$ et $\delta(p_i,q_i) = 1$ si $p_i \neq q_i$.

L'algorithme procède itérativement. A chaque itération, tous les seuils de distance possibles sont d'abord calculés ; on construit pour chaque seuil un graphe non dirigé dont les arcs relient des objets similaires ; pour chaque graphe, on énumère les cliques maximales ; pour chaque regroupement, on calcule une note de qualité ; on choisit les regroupements ayant la meilleure note de qualité et on calcule de nouveaux poids pour toutes les propriétés :

– *étape 1* : on calcule tous les seuils possibles h. Le nombre de seuils l est égal au nombre de distances possibles entre les objets N de l'ensemble T ; $l_{max} = N - (N - 1)/2$, ou inférieur lorsque certaines entités sont équidistantes ;

– *étape 2* : pour chacun de ces seuils, on détermine toutes les paires d'objets similaires selon les définitions [A.1] et [A.3], et on crée un graphe non dirigé dont les arcs relient des objets similaires ;

– *étape 3* : toutes les cliques maximales [A.2] sont calculées pour chacun de ces graphes, aboutit à l regroupements différents ;

– *étape 4* : pour chacun des l regroupements, on calcule une valeur de qualité [1];

– *étape* 5 : on choisit le regroupement le mieux noté par la fonction de qualité, et on calcule de nouveaux poids avec cette fonction[2] ;

– *étape 6* : l'algorithme est répété depuis l'étape 1 pour les nouveaux poids ;

– *étape 7* : l'algorithme se termine quand la valeur de qualité nouvellement choisie est inférieure ou égale à la valeur obtenue à l'itération précédente.

---

1. Voir CAMBOUROPOULOS E., Widmer G., « Automated Motivic Analysis Via Melodic Clustering », *Journal of New Music Research*, 29(4) : 303-318, 2000.
2. *Ibib.*

A.3. L'algorithme FlExPat

A.3.1. *Construction du graphe d'équipollence*

A.3.1.1. *Initialisation*

Créer un graphe vide EG.

A.3.1.2. *Parcours de l'espace des paires de segments*

Parcourir l'espace de toutes les paires de segments légales dans un ordre spécifique, à savoir les positions de segments croissantes combinées avec les longueurs de segments croissantes, comme expliqué dans Rolland[3] (Une paire de segments (s,s') est dite légale quand ni s, ni s', ni la paire elle-même ne sont exclues *de facto* par les contraintes imposées par l'utilisateur ou le programme appelant. Ces contraintes fixent par exemple la différence maximale de longueur entre deux segments, ou le pourcentage maximum de recoupement entre eux).

Pour chaque paire de segments (s,s'), calculer la similitude entre s et s' (en temps constant) et la comparer au seuil numérique SE. Si cette similitude Simil(s,s') est ≥ SE alors :

a) si le sommet correspondant à s n'existe pas encore dans EG, alors le créer. Idem pour s' ;

b) créer une arête entre les sommets correspondant à s et s' dans EG. L'étiqueter avec la valeur Simil(s,s').

A.3.2. *Extraction de sous-graphes (en étoile)*

A.3.2.1. *Initialisation*

Créer une liste vide de sous-graphes en étoile L.

A.3.2.2. *Parcours de l'ensemble des arêtes*

Pour chaque v de EG :

– calculer SimilTotaleVoisinnage(v), qui est la somme des poids (étiquettes) de toutes les arêtes adjacentes à v ;

3. ROLLAND P.Y., « FlExPat: Flexible Extraction of Sequential Patterns », *Proceedings IEEE International Conference on Data Mining (IEEE ICDM'01)*, San Jose, Californie, 29 novembre-2 décembre, 2001.

– si SimilTotaleVoisinnage(v) est supérieur ou égal au seuil numérique T, alors ajouter à la liste L le plus grand sous-graphe en étoile dont le sommet central est v.

A.3.2.3. *Classement et renvoi de la liste résultat*

Classer la liste L par valeurs décroissantes de SimilTotaleVoisinnage (calculées pour les sommets centraux). Renvoyer L.

Index

A

accord 277, 279, 285, 292, 294, 342, 349
acousmatique 122
alarme de réception 178, 186, 188
  de contexte 178
algorithmes 311
Allen 294, 296, 297, 298, 307, 308
Average Magnitude difference function AMDF 47
analyse à court terme 24, 25
  de Fourier 25
  de l'enveloppe spectrale 54
  de l'enveloppe temporelle 30
  des partiels 33
  du fondamental 41
analyse musicale 311
automate 195
  à pile 197, 201-202
  fini 197-202, 204-207, 216, 218-219, 221-225, 227

B

Bach 206-207
BackTalk 351
bande directionnelle 125
bark 101, 402
Bartlett 376
basse chiffrée 345, 348
Binary Format For Scenes (BIFS) 411
binaural 127
Blackman 376
blues 201, 208-211, 215-220
branch & bound 356
Brent 383

C

canon 277, 278, 284
cohérence d'arc 351
combinaison temporelle 275, 292
combinatoire 344
Common Lisp 239
communication 178, 179, 191
composition 339
  assistée par ordinateur (CAO) 231,232
compression 400, 409
concaténation 277-280, 286, 289, 290, 301, 305, 306
cône de confusion 123
contrainte 344
  satisfaction 347
coût 352, 353
CSP 344

D, E

dérive temporelle 192
descripteur 414, 415
distance d'édition 323
écho 185
éditeur de partition 270

effet
de masque 402, 403
de précédence 125
de salle 119
numérique 409
événement 158, 159, 163, 167, 170, 180
expression rationnelle (ou régulière) 198, 202, 222-224

F, G

fenêtre
d'analyse 369, 376
de pondération 27, 28
FlExPat 325, 438
fonctionnalité 270, 277, 280, 297, 299, 300
formants 58
forward checking 348, 354
Fourier 69, 82
fréquence fondamentale 42
Gabor 370
General MIDI 422
gestion du temps 175, 178
gestionnaire de mémoire 177, 181
grammaire 195, 201-203, 208-209, 211-212, 215-221, 270, 299, 300, 301, 305, 306
régulière196-197, 216-217
contextuelle (ou règle) 196, 216

H

Hamming 376
Hann 376
Hanning 378
harmonie 273, 274
règle 341
traité 340
harmonisation 339
Head-Related Transfer Functions (HRTF) 126, 128, 133
holophonie 135, 136, 138, 140, 15
horloge 181

I

incertitude 270, 271
indéterminisme 296, 298
indexation 409
indice
de localisation 123, 126
interaural 123
monoral 123, 125
intensité 402
intéret 356
interpolation parabolique 382

J, L

jazz 195, 203, 208-209, 221-222, 312, 322, 343
Kirchhoff-Helmoltz 137, 140
langages
à objets 233
visuels 233, 234
latence 190, 192
Lex 195, 202, 216, 218-219, 225
logique 295

M

maquette 249-253, 255, 256
Markov 344
masquage 101
Max 234, 235, 263,
mélodie 285, 292, 294
méta-objet 232, 239, 240
métaprogrammation 235, 239-241
méthode du peigne 49
MIDI 157, 420
MIDI Machine Control (MMC) 423
MIDI Time Code (MTC) 422
MIDIFile 420, 423
MidiShare 158
mLAN 432
modéle sinusoïdal 367
Modified Discrete Cosine Transform (MDCT) 404
modulation
d'amplitude 106
de fréquence 107

moteur de recherche 409, 414
musaicing 353
musique tonale 340
MWPP 432

O

occurrence 276, 278, 279, 286, 287, 289
ondelettes 381
Open Sound Control (OSC) 432
OpenMusic 232, 235, 243, 263
opérateur temporel 270
ordonnancement 158, 161, 167, 170
ordonnanceur 158, 163, 170
oscillateur 78, 92
overlap-add 87

P, Q

parsing 424
partiel 68-69, 70, 92, 367
patch 237, 246, 258
pattern 312, 318
prédiction linéaire (LPC) 55
psycho-acoustique 70, 92, 99, 400, 401, 403, 406
pulsation 364
Pulse Code Modulation (PCM) 401
quantification 401
quinte parallèle 342, 347

R, S

répétition 318, 321, 322
représentation temporelle 270
Sample Dump Standard (SDS) 422
ségmentation 312
sémantique opérationnelle 238
séquenceur 188, 270
série, sériel(le) 195, 221-227
seuil d'audibilité 101
similitude 312, 318, 323, 331
SMR 102-103
Society of Motion Picture and Televison Engineers (SMPTE) 422, 429, 432
sous-bande 401, 403, 406, 410
spatialisation 119
spectre 365
splines 97
stéréophonie 133
suivi de partiels 37
  de notes 58
  de partition 61
surround 408
synchronisation 429, 431
synthétiseur 419
système hiérarchique 270, 273, 275, 287, 299, 306

T

tâches temps
temps 429, 430
  réel 158, 161, 179
tessiture 341, 347
tick 429, 430
timbre 74, 125, 415
tonalité 340
Ticks Per Quarter (TPQ) 429
trame 401, 406
transaural 127
transformation sonore 74
transformée de Fourier 363, 368
triton 342

U, W, Y, Z

unscramble 332, 436
workspace 243
Yacc 202
zero-padding 366
ZIPI 432