THESE DE DOCTORAT DE L'UNIVERSITÉ PARIS 6

Spécialité
Informatique

Présentée par
M. Olivier DELERUE

Pour obtenir le grade de
DOCTEUR de l'UNIVERSITÉ PARIS 6

Sujet de la thèse :

Spatialisation du son et programmation par contraintes : le système MusicSpace

Soutenue le 22 Janvier 2004,

Devant le jury composé de :

M. Jean-Pierre Briot	Directeur de thèse
Mme Francesca Rossi	Rapporteur
M. Francis Rousseaux	Rapporteur
M. Jean-François Perrot	Examinateur
M. François Pachet	Examinateur
M. Gérard Assayag	Examinateur

Je remercie les membres du jury pour avoir lu attentivement mes travaux et pour leurs remarques constructives lors de la soutenance et dans leurs rapports.

Je remercie le laboratoire Sony CSL et tous ses chercheurs pour m'avoir accueilli pendant ces longues années.

Merci à Carlos Agon, Jean Carrive, Peter Hanappe et Pierre Roy autant pour leurs conseils avisés que pour leur compagnie sympathique.

Enfin, je remercie vivement François Pachet pour avoir encadré et soutenu ce travail ainsi que pour la confiance dont il a témoigné.

RESUME

Les outils de spatialisation du son actuels, aussi performants soient ils, sont souvent sous-exploités car ils n'offrent pas à leurs utilisateurs de moyens de contrôle pertinents. En particulier, ces systèmes ne permettent pas d'exprimer les relations qui existent entre les différentes sources qui composent une scène sonore, relations pourtant fondamentales à la cohérence et pertinence du résultat sonore.

Nous proposons, pour remédier à ce manque, d'utiliser la programmation par contrainte, paradigme qui fournit non seulement des moyens adaptés pour décrire des relations arbitraires entre les variables d'un système, mais également des méthodes de calculs permettant de résoudre les problèmes posés.

Ces concepts sont intégrés au sein d'un prototype logiciel, MusicSpace, regroupant un moteur de contraintes ad hoc, une interface de représentation et contrôle de la scène sonore, ainsi que des moyens de communication permettant le contrôle de spatialisateurs existants.

Les éléments d'argumentation initiaux qui motivaient notre démarche sont finalement mis en application et confirmés au sein de la phase d'expérimentation qui met en scène notre système dans une panoplie de situations d'utilisation.

ABSTRACT

Tools for sound spatialization, although they might be nowadays very efficient, are often misused because they don't provide their users with consistent and meaningful means for controlling the sound scene. Particularly, it is not possible to specify and maintain the important relationships that exist between the different sound sources of an auditory scene and that are necessary to ensure the consistency of the result.

We propose, in order to address this problem, the use of constraint programming which provides means for establishing arbitrary relations between the variables of a system, as well as algorithmic means for solving the problems that are being posed.

These ideas are integrated in our software prototype, MusicSpace, which gathers together a constraint solving toolkit, a graphical user interface for representing and editing the auditory scene and communication means for controlling existing spatialization systems.

The initial motivations for this research are applied and confirmed within the experimentation phase of our system, which consists in a number of case studies.

TABLE DES MATIÈRES

Introduction

Grâce à Edison, puis à bien d'autres successeurs, nous pouvons enregistrer et reproduire le son et la musique sur des supports, et réécouter ces enregistrements à loisir. Néanmoins, la production de musique ne se borne pas à un simple enregistrement d'une source sonore. Le mot même de « production » sous-entend toute une chaîne de traitement de ces sources, de transformations, qui est devenue aujourd'hui centrale dans le processus de fabrication de la musique. En particulier, le mixage de sources sonores joue un rôle prépondérant en musique populaire (Pop, Rock, etc.) pour laquelle les différents instruments sont traditionnellement enregistrés séparément, et mixés ensuite par un - ou plusieurs - ingénieur du son. L'art de cet ingénieur, son savoir-faire ses connaissances, mais aussi ses parti pris esthétiques sont pour beaucoup dans le résultat final. Dans le contexte d'applications interactives où l'on va laisser l'utilisateur agir sur l'emplacement de sources, on doit donc se demander en premier lieu ce qui constitue un « bon » mixage.

La cohérence du résultat sonore repose en grande partie sur les relations temporelles et spatiales qu'entretiennent les sources sonores entre elles d'une part et avec l'auditeur d'autre part. Afin d'assurer leur crédibilité, les systèmes informatiques actuels qui proposent de recréer artificiellement des environnements sonores se doivent donc de tenir compte et de respecter ces propriétés.

Le domaine temporel a longtemps prédominé sur les aspects relatifs à l'espace, mais les questions techniques qu'il soulevait, la synchronisation par exemple, ont trouvé des réponses fiables. Aujourd'hui c'est au tour du domaine spatial de faire l'objet d'études approfondies.

Ainsi, de nombreux systèmes de spatialisation commencent à voir le jour : ils permettent à leurs utilisateurs d'ajuster librement les paramètres spatiaux des sources sonores. En fonction de l'acuité du système de spatialisation utilisé, tout une gamme de paramètres est proposée, allant du simple réglage panoramique, dosage d'équilibre d'une source entre des haut-parleurs gauche et droit, jusqu'à des paramètres de composition géométrique, de position, d'orientation ou de vitesse.

Malheureusement, la plupart de ces systèmes ne permettent pas de tenir compte des relations spatiales qui lient les sources entre elles : l'ajustement des paramètres reste empirique et, la majeur partie du temps, figé de manière statique à une configuration arbitraire.

Nous défendons l'idée qu'il est possible de faire la part entre certains aspects fondamentaux de

la spatialisation que seul un spécialiste, un ingénieur du son est en mesure d'exprimer, et une composante arbitraire, que l'utilisateur devrait pouvoir adapter à son goût.

Nous illustrons notre propos par le projet MusicSpace, qui permet de modéliser des connaissances sur la spatialisation, sous la forme de relations établies entre les sources et l'auditeur et offre un espace d'exploration réduit dans lequel les manipulations obtenues font sens. MusicSpace se compose d'une part d'une interface graphique de visualisation et manipulation de la scène sonore et d'autre part d'un moteur de contraintes permettant de maintenir des relations sur les différentes sources de la scène. Notre système propose par ailleurs des moyens de définir rapidement et facilement ces contraintes : l'utilisateur ajoute et retire des contraintes du système en cours d'exécution et peut donc ainsi en évaluer directement la convenance sur le résultat sonore et la cohérence de la scène. Pour cela, notre système se connecte de plusieurs manières à un ensemble de composants matériels ou logiciel capables de produire un rendu en temps réel de la scène sonore spatialisée.

Dans un premier temps, nous rapportons dans ce document l'étude de différents domaines auxquels notre projet se rapporte, à commencer par la spatialisation du son : nous tentons d'en dresser un panorama de l'état actuel tant au niveau des motivations que des différentes techniques disponibles en regard des différentes applications vers lesquelles elles se tournent spécifiquement. A cette occasion, nous observerons l'ensemble des paramètres que chacun des systèmes, chacune des techniques, propose comme moyens de contrôle sur le résultat sonore. Cette section se termine par l'observation des systèmes d'interfaçage des systèmes de spatialisation, couches logicielles fondamentales puisqu'elles tentent à standardiser l'ensemble des paramètres de contrôles de scènes sonores virtuelles et permettent de s'affranchir des spécificités des techniques sous-jacentes employées. s

Puis nous ferons état de quelques-uns des rares travaux de recherche qui prennent place dans le domaine central de notre étude : les interfaces de contrôle de la spatialisation du son. Ce sera en particulier l'occasion d'observer tout spécifiquement la « table de mixage » ainsi que les concepts d'opérabilité qu'elle met œuvre.

Pour finir, nous nous concentrerons sur la « programmation par contrainte », paradigme de la résolution de problèmes, dont l'utilisation est non seulement la clef de voûte de notre système, mais en fait également toute son originalité. Après un rappel des objectifs et des principes propres à la programmation par contrainte, nous nous efforcerons à en distinguer les différentes démarches et nous orienterons vers celle qui se prête le mieux à nos besoins, celle des algorithmes de propagation locale.

Au sein de la deuxième partie de ce document, nous décrivons notre système, MusicSpace, une interface de contrôle de la spatialisation du son, qui, au moyen de contraintes, permet de définir un ensemble de relations portant sur les différentes sources qui composent une scène sonore, afin d'aboutir à un environnement de contrôle de la spatialisation cohérent, qui fait sens, et spécialement adapté à la scène sonore. Nous détaillons dans cette partie, les aspects techniques liés à l'implémentation de notre système, et en particulier, dressons l'inventaire des contraintes qui y sont intégrées.

Finalement, la troisième partie de ce document est consacrée à l'expérimentation que nous avons pu effectuer avec notre système. Nous y exposons un ensemble de cas types de mise en situation qui permettent au lecteur de se construire une idée concrète de différentes

applications possibles de notre systèmes ainsi que des problèmes spécifiques auxquels celui-ci est confronté dans chacune de ces mises en œuvre. Nous y décrivons également les travaux de recherche de Nicolas Deflache, qui, à l'occasion du projet de recherche de fin d'études de sa « formation supérieure aux métiers du son », a accepté de consacrer du temps à l'évaluation de notre système, MusicSpace, dans un contexte professionnel de mixage.

Nous tâcherons en conclusion, de préciser l'apport de ce travail en regard des autres travaux existants dans le domaine du contrôle de la spatialisation. Nous tâcherons d'effectuer une synthèse des points forts autant que des faiblesses de notre projet afin d'en dégager un ensemble de pistes de recherches pour des travaux ultérieurs complémentaires.

Pour finir, le lecteur trouvera en annexe de ce document la description d'un ensemble de points techniques qui ont fait l'objet d'efforts particuliers dans cette étude mais qui n'occupent pas une place centrale au sein de notre axe de recherches.

Première Partie
État de l'art

Chapitre 1
Spatialisation du Son

> Spatialisation :
>
> MUSIQUE - (n.f.) En concert, répartition sélective du son dans l'espace tridimensionnel, au moyen de divers dispositifs d'amplification utilisant une quantité variable de haut-parleurs, ainsi que des accès (console, systèmes informatisés, etc.) permettant éventuellement d'agir en temps réel sur différents paramètres sonores : l'intensité, la localisation, le mouvement, la coloration spectrale, etc.
>
> Dictionnaire des arts médiatiques
>
>

1.1 Motivations et Historique

« On peut entrevoir un orchestre nombreux s'augmentant encore du concours de la voix humaine […]. Par cela même, la possibilité d'une

> musique construite spécialement pour le « plein air », toute en grandes lignes, en hardiesses vocales et instrumentales qui joueraient et planeraient sur la cime des arbres dans la lumière de l'air libre. »
>
> Claude Debussy, 1903

La spatialisation, ou prise en compte de l'espace comme paramètre de jeu dans la composition musicale a toujours été au centre des préoccupations chez les compositeurs. Si cette volonté apparaît sous forme assez onirique pour Debussy (voir citation ci-dessus, [Debussy, 1903]) qui en 1903 parle déjà de « musique planante », elle l'est de façon bien plus concrète et expérimentale à différents moments de l'histoire lorsque par exemple à la création de « Fantasia » de Walt Disney, en 1940, Stokowski dirige un orchestre scindé en quatre parties ou encore lors de la création de « Carré » de Stockhausen, quand quatre orchestres et quatre chœurs arrangés en carré jouent pour un public disposé au centre.

Ces exemples restent relativement anecdotiques en particulier en raison des difficultés techniques que représente la mise en place de tels événements artistiques. La technologie est cependant venue au secours des compositeurs en proposant au fil des années des moyens de plus en plus « abordables » offrant toute une série de d'approches pour aborder le paramètre spatial.

Une première avancée fondamentale a été l'invention puis le perfectionnement des techniques d'enregistrement et de restitution du son permettant en outre l'enregistrement multipiste, étape indispensable à la l'apparition de la notion de « source sonore ». L'invention initiale est celle de Thomas Edison, en 1877, le Gramophone, la première machine à enregistrer de l'histoire qui permettait de reproduire deux minutes de vibrations sonores sur des cylindres d'étain tout d'abord puis de cire par la suite. Vient ensuite l'invention du magnétophone en 1935 par la firme allemande AEG, celle du 33 tours en 1948 par la compagnie Columbia et du 45 tours en 1949 par RCA, et finalement l'apparition du premier magnétophone multipistes, en 1952, le « tripistes » qui est employé par Olivier Messiaen au GRM. Le nombre de pistes que les magnétophones sauront enregistrer et reproduire simultanément va ensuite croître progressivement, allant de 8 à 16, 32 et même 48 pistes, jusqu'à ce que les techniques d'enregistrements informatique apparaissent pour introduire la notion de magnétophone virtuel : le nombre de pistes n'est alors plus conditionné physiquement par la largeur de la bande magnétique ou de la tête d'enregistrement mais simplement par les limites de la puissance de calcul de la machine et peut donc être a priori augmenté à volonté, à mesure que les machines deviennent plus puissantes.

Une autre avancée technique importante est liée aux systèmes de diffusion : la restitution de la musique monophonique originale sur un haut-parleur a été remplacée par la stéréophonie, la quadraphonie et aujourd'hui des dispositifs proches de ceux du cinéma tels que le 5.1 par exemple. Par ailleurs le développement de systèmes permettant la diffusion sur un nombre important de haut-parleurs tel que l'Acousmonium du GRM apparu en 1973, le « GMEBaphone » de Bourges, ou l'orchestre de haut-parleurs du BEAST (Birmingham) ont largement contribué à entretenir une place essentielle à l'écriture de l'espace dans le travail des compositeurs. Ces outils originalement empiriques sont progressivement relayés par des systèmes technologiquement plus avancés comme par exemple celui développé par le Bell

Telephone Laboratories, en 1982, utilisant un rideau de microphones faisant face à une source sonore, et alimentant un rideau similaire de haut-parleurs de sorte à reconstruire le front d'onde correspondant.

Finalement, les techniques informatiques ont permis de mettre en commun les moyens de reproduction et les moyens de diffusion en y ajoutant un niveau de traitement permettant de servir d'interface entre des sources sonores anéchoïques et leur mise en espace au travers d'un système de diffusion arbitraire. Le pionnier dans ce type de recherches est probablement John Chowning qui simule en 1971 par programme informatique des mouvements rapides des sources sonores. Il réalise aussi un dispositif analogique où cette spatialisation est commandée par un manche à balai (joystick) (c.f. [Chowning, 1971]). Depuis, de nombreuses recherches informatiques ont contribué à améliorer ces méthodes pour rapprocher les résultats des systèmes de diffusion le plus possible de la perception auditive humaine : des outils relativement perfectionnés en résultent, tels que le Spatialisateur IRCAM (IRCAM & Espaces Nouveaux) par exemple, apparu en 1994, qui calcule les modifications nécessaires à apporter au signal sonore pour en produire la spatialisation par un système de restitution arbitraire (ensemble de haut-parleurs, casque,…).

A l'heure actuelle, les besoins en spatialisation sonore ont largement dépassé les considérations et ambitions esthétiques originales des compositeurs en s'invitant dans des domaines applicatifs nouveaux où ils sont nécessaires par essence. Ces domaines sont par exemple les réalités virtuelles où un effort particulier est porté au niveau de la qualité de chacun des différents rendus, visuel, sonore, et parfois même proprioceptif afin de convaincre l'utilisateur d'une certaine cohérence entre les différents médias, porteuse de la sensation d'immersion recherchée qui plonge celui-ci dans l'univers artificiel qu'il explore. L'enjeu est encore plus grand dans le cadre des réalités augmentées où le monde réel est « augmenté » d'éléments virtuels visuels ou sonores : l'objectif est une fois de plus de réussir à immerger l'utilisateur mais cette fois-ci en le convainquant de la matérialisation d'éléments virtuels au sein de son univers physique (voir [Delerue & Warusfel, 2002]).

Dans les sections suivantes nous rappelons les origines physiologiques de la perception auditive spatiale, avant de passer, au paragraphe 1.3, à l'étude des différents systèmes et procédés qui permettent de simuler cette sensation de localisation spatiale.

1.2 Principe

Les origines physiologiques de la perception auditive spatiale ont fait l'objet d'un nombre considérable d'études et de rapports. Nous en décrivons ici les concepts de base et invitons le lecteur désireux d'approfondir le sujet à se reporter par exemple à [Burgess, 1992], [Blauert, 1999] ou encore [Begault, 2000].

En milieu anéchoïque, les indices permettant la localisation sont essentiellement les différences en temps, en intensité et en timbre entre les signaux émis par une source « ponctuelle » reçus aux deux oreilles d'un auditeur. Les différences de temps sont dues simplement à l'écartement des oreilles, tandis que les différences en timbre et en intensité sont provoquées par l'effet de masquage et d'absorption que produit la tête, la peau, les cheveux… sur le signal sonore.

Burgess ([Burgess, 1992]) définit 8 types d'indices jouant une place particulièrement importante dans le cadre de la localisation en direction et en distance : l'ITD (Interaural Delay Time), le

masquage de la tête (Head Shadow), la réponse en fréquence de l'oreille externe (Pinna Response), les échos produits par le buste, les mouvements de la tête, la vue, les échos précoces provenant de l'environnement ainsi que la réverbération. Les quatre premiers forment ce que l'on définit comme HRTF (Head Related Transfert Function) et agissent de manière importante sur la perception de localisation des sources sonores en direction. Pour la perception en distance, les paramètres prépondérants sont, en milieu anéchoïque, la décroissance de l'amplitude de la pression acoustique suivant la distance, ainsi que l'atténuation des hautes fréquences due à la l'absorption atmosphérique. Ces critères sont nécessaires mais pas suffisants : en particulier, ils ne permettent pas de faire de distinction entre un son de forte intensité émis par une source distante de l'auditeur avec un son identique mais de faible intensité émis cette fois part une source à proximité de l'auditeur. Lorsque le milieu dans lequel l'auditeur se trouve est réverbérant, les rapports entre l'énergie du signal direct et celle du signal réverbéré s'ajoute comme un indice supplémentaire d'importance relativement élevée.

D'autres facteurs entrent également en jeu comme l'aspect visuel par exemple. Mais il s'agit également de la nature de la source sonore et de la potentialité de l'objet qui la produit à tournoyer au dessus de l'auditeur ou au contraire à rester irrémédiablement accroché au sol. Des études perceptives ont par ailleurs montré que la nature même de la source sonore peut influencer la localisation spatiale lors de l'écoute dans le plan médian : par exemple ([Blauert, 1999]) une source composée d'une voix humaine diffusée en face a tendance à être effectivement perçue vers l'avant lorsque cette voix est connue du sujet. En revanche, lorsque cette voix est inconnue du sujet l'ensemble des stimuli auditifs sont perçus en dehors du champ de vision, soit au dessus de la tête, soit comme provenant de l'arrière.

Ces facteurs, relativement complexes à prendre en compte sont généralement laissés de coté dans les systèmes de simulation de localisation sonore : en particulier ils demandent des connaissances a priori sur le contenu sonore à spatialiser. Seuls les paramètres physiques systématiques, ceux qui ne dépendent pas des sources sonores sont pris en compte. Nous dressons dans les prochains paragraphes la description des systèmes, techniques et procédés de spatialisation les plus couramment rencontrés.

1.3 Systèmes, Techniques et Procédés

La spatialisation du son se présente aujourd'hui comme un domaine de recherche établi et a déjà fait l'objet d'une quantité considérable d'études tant sur la perception que sur la mise au point de méthodes permettant de la synthétiser. Le résultat en est un nombre conséquent de systèmes de traitement du signal permettant de transformer un signal sonore anéchoïque pour lui attribuer des qualités spatiales tant en orientation qu'en distance ainsi que simuler le comportement sonore d'un environnement physique par le biais, par exemple, de réverbération.

La spatialisation consiste en fait en deux tâches distinguables : l'une consiste à produire à partir du son anéchoïque, l'ensemble de la réponse impulsionnelle correspondant à un environnement donné : cette réponse est couramment divisée en 4 tranches temporelles distinctes représentant respectivement le son direct, les premières réflexions, les réflexions tardives et la réverbération (voir Figure 1). Ces signaux retardés et filtrés du son original permettent de simuler la perception de distance qui, comme nous l'avons vu au paragraphe précédent, s'exprime en première approximation comme le rapport de l'énergie du son direct

par l'énergie du signal réverbéré.

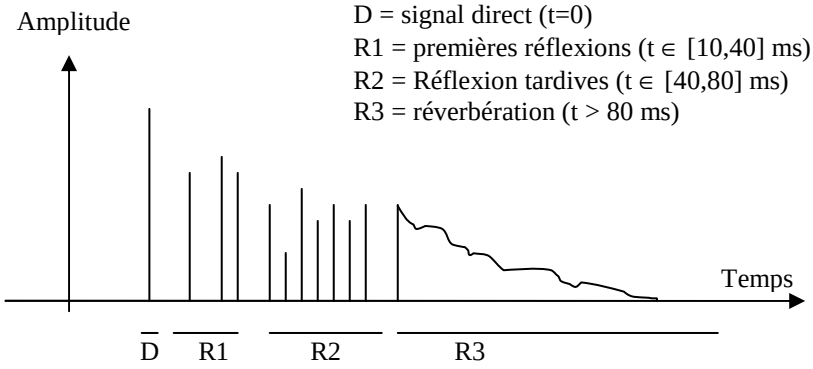


Figure 1 : décomposition de la réponse impulsionnelle en quatre groupes temporels

La seconde tâche nécessaire à la spatialisation consiste à attribuer à un signal sonore une information d'orientation pour l'auditeur : cette information de localisation est générée à partir des d'indices de différence interaurale d'amplitude, différences de temps et de filtrage. Les deux tâches se combinent alors pour attribuer à chacun des signaux entrant dans la composition de la réponse impulsionnelle la localisation spatiale nécessaire.

Les systèmes rencontrés s'intéressent donc parfois à l'une ou à l'autre tâche ou encore, aux deux, pour les plus complets, et ce dans un contexte de restitution et diffusion sonore bien précis. D'autres, au contraire, – en particulier ceux qui génèrent la spatialisation au moyen d'une modélisation physique – peuvent adopter un point de vue complètement différent et ne pas distinguer ces cas. Par ailleurs, un paramètre qui démultiplie considérablement les prototypes est le système de reproduction envisagé : chaque cas, haut-parleurs stéréos, haut-parleurs quadriphoniques, casque, système 5.1, réseaux de haut-parleurs… génère sont propre procédé. Il en résulte donc un nombre important de méthodes, et de systèmes.

Nous présentons dans les sections suivantes de ce document un éventail des techniques les plus couramment utilisées, des plus empiriques aux plus sophistiquées en gardant comme préoccupation centrale l'observation de leurs paramètres de contrôle : chaque technique de spatialisation s'accompagne en effet d'un jeu de paramètres spécifiques et génère donc des modalités de contrôle qui lui est propre.

1.3.1 Approche Physique

L'approche physique consiste à modéliser la scène sonore à partir de paramètres géométriques d'une part, et physique de l'autre afin de décrire le plus précisément possible les positions des parois, leurs propriétés acoustiques, la position des sources, éventuellement leurs modèle de rayonnement, ainsi que la position du point d'écoute pour calculer le résultat sonore au moyen d'algorithmes tels que le calcul de sources images, le lancé de rayon ou l'application de modèles de radiosité ([Cerveau, 1999]). Dans un tel cas, il n'y a pas réellement séparation des tâches de simulation acoustique et de localisation des sources : chaque réflexion du signal contre des parois apporte sa contribution au signal généré, le tout formant le rendu spatialisé.

Cette approche n'est généralement choisie que dans le cadre des applications d'acoustique

prévisionnelle d'une part parce que les algorithmes correspondant sont extrêmement coûteux, et d'autre part parce qu'elle ne propose pas à son utilisateur de moyens de contrôles accessibles et significatifs. En effet, outre les positions et orientations relatives des sources et du point d'écoute qui, nous le verrons apparaissent comme paramètre de contrôle dans la quasi-totalité des méthodes de spatialisation, les modalités de contrôle de ce type de spatialisation s'expriment au travers de la définition géométrique de l'environnement physique ainsi que des différents coefficients (de réflexion par exemple) des éléments qui le composent. Son utilisation est donc particulièrement indiquée lorsqu'il est nécessaire de construire un rendu sonoreproche d'une certaine forme de réalité, et moins adaptée lorsque la spatialisation est utilisée comme un « effet » en particulier en musique.

1.3.2 Procédés de pan-pot, stéréophonie conventionnelle.

Cette méthode présuppose un mode de diffusion sur deux canaux et consiste, pour une source monophonique donnée, à doser la quantité de signal qui sera transmis à chacun des canaux. Cette différence d'intensité du signal émis par les deux haut-parleurs permet – lorsque l'auditeur n'est pas trop mal situé par rapport à ceux-ci – d'approcher avec un certain degré d'approximation la différence interaurale de niveau perçue naturellement et ainsi de donner une sensation de variation de l'azimut de la source sonore. Ce procédé sert donc au positionnement d'une source dans une image sonore située entre les deux haut-parleurs uniquement et ne simule a priori aucun effet de salle.

Bien que très rudimentaire, ce procédé est d'importance capitale puisque, implanté de manière directe sur toutes les consoles de mixage (c.f. section 2.1 de ce document) en tant que réglage de « panpot », il est utilisé en la quasi-totalité des enregistrements sur disque compact disponibles à l'heure actuelle. En dépit de sa pauvreté en précision, il permet toutefois de positionner les sources à différents points de la scène sonore de sorte à produire une image stéréophonique pleine et équilibrée. Par ailleurs, il possède l'avantage immense de résister à une réduction monophonique, impérative pour toute diffusion radiophonique, ce qui n'est pas le cas des procédés faisant intervenir les différences interaurales de temps, dont les effets de phases peuvent en cas de réduction monophonique avoir des conséquences désastreuses sur le résultat sonore.

1.3.3 Emission par des sources, orchestres de haut-parleurs

Cette idée de la spatialisation vise à considérer l'espace comme paramètre soit de composition, soit de jeu instrumental, au moyen d'un ensemble de haut-parleurs de colorations harmoniques et position spatiales différentes de sorte à démultiplier les deux canaux stéréophoniques originaux placés à l'entrée d'une console de mixage lors de la diffusion d'œuvres électroacoustiques. L'exemple le plus célèbre d'un tel dispositif est probablement l'Acousmonium, conçu par François Bayle (INA-GRM) et réalisé par Jean-Claude Lallemand en 1974. L'acousmonium est constitué d'un nombre très variable de haut-parleurs (de quelques paires à plus d'une centaine) ou « projecteurs de sons » (Bayle), de caractéristiques et de colorations harmoniques différentes, contrôlés par un « directeur du son » à partir d'une console spéciale.

Chaque piste sonore de l'enregistrement à diffuser est assignée à des sorties directes de la console de mixage et commandée individuellement par autant de potentiomètres et de

systèmes d'égalisation. Ces canaux correspondent chacun à un ou plusieurs haut-parleurs, placés à des endroits déterminés par les conditions acoustiques du lieu et par les stratégies artistiques choisies pour la « mise en espace » des oeuvres.

Cette technique de spatialisation est totalement empirique et repose sur une technicité et un savoir faire de la personne qui diffuse la pièce. Elle repose également sur un nombre de haut-parleurs très important puisque en théorie, un haut-parleur est utilisé pour chaque position rendue. Les paramètres de contrôle sont typiquement ceux de la console de mixage : ils ne sont donc pas normalisés puisqu'ils varient d'un système à l'autre et il est difficile de formaliser

1.3.4 Synthèse binaurale

L'idée sous-jacente de la synthèse binaurale est relativement simple : elle consiste à reconstruire, à partir d'un enregistrement monophonique et anéchoïque initial, les signaux gauche et droit qui parviendraient aux oreilles si une source émettant le signal initial était placée à une distance et sous une orientation données par rapport à l'auditeur. Le résultat est transmis à celui-ci au moyen d'un casque de telle sorte que chacun des signaux gauche et droit construits sont remis aux oreilles correspondantes. Les principes de la spatialisation binaurale sont connus depuis un certain temps puisque déjà, en 1920, Harvey Fletcher conçoit un système binaural fonctionnant sur casque. En revanche sa simulation par des procédés de traitement du signal est plus récente puisque les convolutions nécessaires dans le cas général sont coûteuses en puissance de calcul et nécessitent l'utilisation de microprocesseurs récents.

Si la simulation binaurale peut produire des résultats saisissants de localisation sonore tridimensionnelle, un des désavantages importants de l'écoute au casque est l'absence d'image frontale : les sons situés en face de l'auditeur ont la mauvaise tendance à être perçus comme provenant de l'intérieur ou éventuellement au dessus de la tête. Par ailleurs la synthèse de la spatialisation par des techniques binaurales requiert si possible l'utilisation d'un système de suivi permettant de compenser les mouvement de la tête de l'auditeur : en effet, l'image spatiale créée se dégrade rapidement en cohérence lorsque l'auditeur tourne la tête, par exemple, (un des réflexes connus chez l'humain et l'animal pour affiner l'estimation de localisation d'une source sonore) et que la scène auditive n'est pas adaptée en conséquences. Un tel dispositif nécessite des manipulations du signal en temps-réel relativement coûteuses en puissance de calcul. Le lecteur souhaitant d'avantage d'informations sur les techniques binaurales et en particulier l'encodage multicanal (qui permet d'économiser de la puissance de calcul lorsque le nombre de sources sonores devient important) pourra se reporter à [Larcher, 2001].

La synthèse binaurale est un procédé de localisation de sources sonores : elle apporte à un signal anéchoïque des informations de localisation en élévation et azimut. Ces deux paramètres sont donc l'essentiel du contrôle que permet ce procédé, hormis les données nécessaires au filtrage binaural, exprimé soit sous forme d'un ensemble de réponses impulsionnelles soit sous forme de données morphologique relatives à la taille et forme de la tête de l'auditeur.

1.3.5 Reproduction transaurale

La reproduction transaurale est un procédé visant à étendre la validité de la synthèse binaurale à un système de diffusion sur deux haut-parleurs : il consiste en l'annulation des signaux croisés, e.g. le signal émis par le haut-parleur gauche provenant à l'oreille droite de l'auditeur, et, inversement, le signal émis par le haut-parleur droit et provenant à l'oreille gauche.

Les résultats produits par ce type d'approche sont conséquents, en particulier au niveau de l'image sonore frontale beaucoup mieux définie que dans le cas de la synthèse binaurale en raison du casque. En revanche, l'inconvénient de cette technique est la précision avec laquelle l'auditeur doit être positionné par rapport aux haut-parleurs pour fonctionner. De plus les mouvements de la tête détruisent l'image spatiale.

En conclusion cette méthode convient particulièrement à des situations individuelles de type multimédia par exemple, dans laquelle l'auditeur reste relativement statique et conserve généralement son regard centré sur un système de rendu visuel, ce qui limite les mouvements de rotation de la tête. La reproduction transaurale n'apporte pas de paramètres de contrôle supplémentaire significatifs pour l'utilisateur.

1.3.6 Approche ambisonique

L'approche ambisonique ([Gerzon, 1972]) consiste en un procédé d'encodage et de décodage de la scène sonore : il est fondé sur la définition locale du champ acoustique suivant des harmoniques sphériques d'ordre arbitraire, en un point de l'espace correspondant à la position de l'auditeur. Dans son format traditionnel (le B-format qui correspond à la décomposition en harmoniques sphériques d'ordre 1), ce champ est représenté par une composante omnidirectionnelle W et trois composantes bidirectionnelles X, Y, et Z, orthogonales. Le lecteur intéressé pourra se référer à la thèse de Jérome Daniel ([Daniel, 2000]) pour d'avantage d'informations sur ce procédé où à [Malham & Myatt, 1995] qui propose une implantation en langage C-sound.

Une originalité intéressante du procédé ambisonique réside dans les possibilités de contrôle que ce format offre avant décodage. S'il n'est pas possible de manipuler indépendamment les différentes sources sonores qui composent la scène, le B-format autorise néanmoins une série de transformations peu coûteuses concernant l'ensemble de la scène telles que les rotations suivant les différents axes, ainsi qu'une opération de « focus » permettant de déformer l'espace sonore visant à privilégier une direction donnée par rapport aux autres.

1.3.7 Procédés Holophoniques et Antennes

L'holophonie est le pendant auditif de l'holographie. Le procédé consiste à reproduire le champ acoustique en tout point d'une zone dite de restitution au moyen de matrices de haut-parleurs idéalement « entourant » cette zone. La méthode s'appuie sur les propriétés énoncées dans le principe de Huygens : celui-ci stipule que le front d'onde rayonné par une source dite primaire se comporte comme une distribution de sources dites secondaires. Le champ induit en aval d'un front d'onde peut donc être vu comme le champ rayonné par une infinité de sources réparties le long de ce front d'onde. Les sources secondaires se substituent ainsi parfaitement aux sources primaires ([DeVries & Boone, 1999] et [Nicol, 1999]).

Un des défauts de cette méthode est lié à la résolution spatiale et a son incidence sur la bande passante correctement restituée. En revanche un de ses aspects singuliers est que la notion d'auditeur ou de point d'écoute – telle qu'on la trouve généralement dans les interfaces de contrôle – est supprimée pour être remplacée par une zone d'écoute au sein de laquelle le champ acoustique est « parfaitement » restitué et où l'auditeur peut évoluer librement. Les coordonnées des sources sont alors représentées de manière absolue, dans un repère où figure également la zone d'écoute.

1.3.8 VBAP : Vector Base Amplitude Panning.

Le « Vector Base Amplitude Panning » est une méthode de spatialisation développée par Ville Pulki ([Pulkki, 1997], [Pulkki, 2001], [Pulkki & Al, 1999]) permettant elle aussi le positionnement de sources sonores virtuelles au moyen d'un ensemble de haut-parleurs. Un des aspects intéressants de ce projet est que le nombre de haut-parleurs utilisés peut être variable et que ceux-ci peuvent être disposés de manière arbitraire dans un plan ou dans un volume.

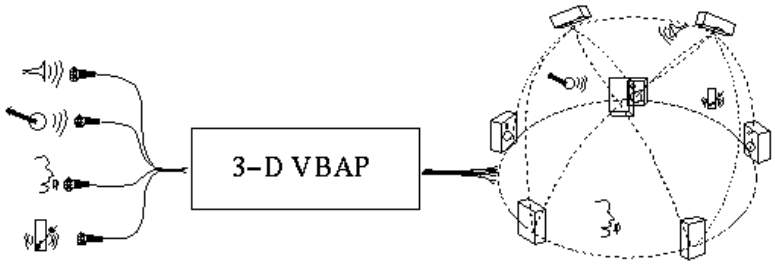


Figure 2 : traitement de sources sonores distinctes dans VBAP

Le principe de VBAP passe par une reformulation plus générale de la méthode « panpot d'intensité » traditionnelle (voir section 1.3.2) dans laquelle deux haut-parleurs délivrent un signal cohérent avec des amplitudes différentes. Le procédé existe sous deux formes : une forme bidimensionnelle dans laquelle il est supposé que les haut-parleurs sont tous disposés dans un plan horizontal, et une forma

Dans sa formulation en deux dimensions (2-D VBAP), un vecteur-source est considéré comme étant la somme pondérée de vecteurs-haut-parleur. Sur la Figure 3 (gauche) les poids g_1 et g_2 sont affectés aux vecteurs-haut-parleur l_1 et l_2 pour construire le vecteur-source p. Il a été montré ([Pulkki & Al, 1999]) qu'après normalisation, les poids obtenus peuvent être utilisé comme des gains en intensité et que le résultat sonore rejoint la loi tangente largement utilisée en informatique musicale (voir [Moore, 1990]).

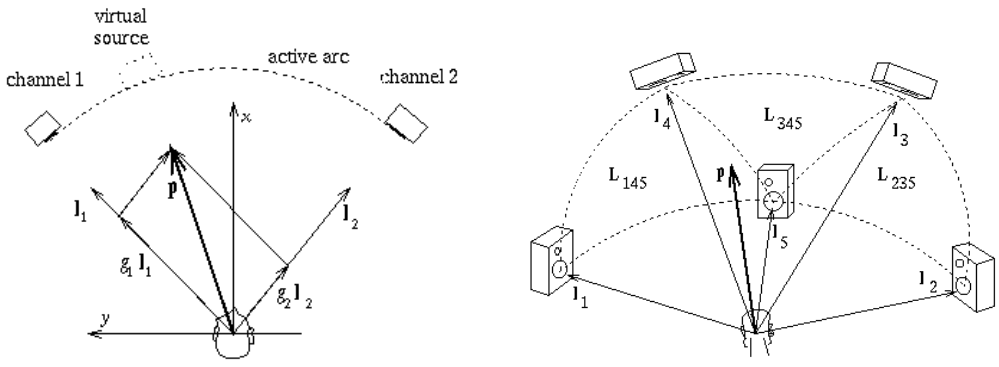


Figure 3 : VBAP, formulation 2D (gauche) et 3D (droite)

Par extension, la version tridimensionnelle du projet (3D-VBAP) s'intéresse à des triplets de

haut-parleurs. Ces triplets définissent des zones à l'intérieur desquelles des sources virtuelles peuvent être localisées, en utilisant un calcul de gain similaire à celui de la version bidimensionnelle. Quel que soit le nombre de haut-parleurs disponibles, chaque source n'est donc émise que par un nombre limité (1, 2 ou 3) de haut-parleurs. En revanche, davantage de haut-parleurs peuvent être utilisés au même moment pour localiser des signaux sonores entrant dans la composition de l'effet de salle, des premières réflexions ou du champ diffus.

Le projet lui même existe sous la forme d'une extension pour le logiciel C-sound, ou d'une collection d'objets pour l'environnement MAX ([Puckette, 1988]) effectuant le travail de traitement de signal : aucune interface graphique n'a été conçue spécifiquement pour le contrôle dans ce travail.

Une variation de l'approche VBAP est proposée par Jean-Marie Pernaux , Patrick Boussard et Jean-Marc Jot, « VBIP » (Vector Based Intensity Panning), et consiste en une combinaison de la technique dite de Vector Based Panning et du procédé d'encodage Ambisonic afin de satisfaire davantage les critères de localisation dans les hautes fréquences. Le lecteur trouvera davantage d'informations à ce sujet dans ([Pernaux & Al, 1998]).

1.3.9 Le Spatialisateur IRCAM

La librairie Spatialisateur de l'IRCAM ([Jot, 1999]) n'est pas une méthode ou un procédé mais un environnement de spatialisation qui permet de combiner les différentes approches vues précédemment pour réaliser entièrement la spatialisation de sources sonores (localisation spatiale et effet de salle) dans un contexte temps réel.

Ce projet se distingue des autres systèmes par son esprit modulaire qui permet de combiner la plupart des différentes approches de la spatialisation : le système peut ainsi être adapté spécifiquement à une situation d'utilisation, tenant compte de contraintes liées au dispositif de reproduction utilisé, à la puissance de calcul disponible et au nombre de sources sonores à spatialiser simultanément. Il est fondé sur la décomposition de la réponse impulsionnelle en 4 sections temporelles comme présenté au paragraphe 1.3 et la véhicule au moyen d'une représentation interne sur 7 canaux audio :

- Un canal de son direct, éventuellement filtré et atténué pour rendre compte de l'orientation de la source et de sa directivité.

- Deux canaux de premières réflexions. Ces réflexion interviennent généralement dans la zone 10 – 40 ms après l'apparition du son direct et correspondent à ses premières réflexions sur les parois. Chacun de ces canaux contiennent quatre copies du son direct, retardé et sont généralement spatialisées à +/- 30° de part et d'autre de la source sonore.

- 4 canaux mélangeant les réflexions tardives et la réverbération.

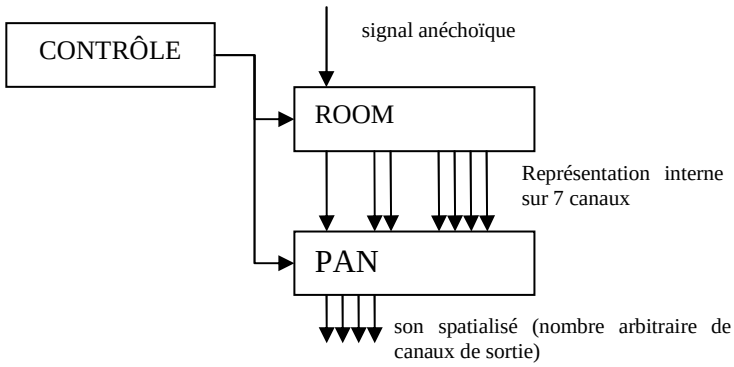


Figure 4 : schéma de base du Spatialisateur IRCAM

Dans un deuxième temps, ces canaux sont traités pour recevoir une information de localisation : c'est le travail du module « pan » (Figure 4) : un format commun de diffusion est décidé (la quadriphonie par exemple) et chacun des sept signaux construits au sein du module « Room » sont encodés dans ce format avec les informations de localisation nécessaires avant d'être mixés vers les sorties principales. Généralement, le son direct est traité avec la précision maximale pour la localisation en azimut comme en élévation. Puis au fur et à mesure que l'on progresse dans la réponse impulsionnelle, des techniques moins coûteuses sont employées puisque les indices de positionnement angulaire sur la réverbération par exemple ne contribuent que peu dans la perception de localisation.

Les canaux correspondant aux réflexions précoces correspondent à huit copies retardées du son original, groupées par 4, et spatialisées symétriquement de part et d'autre du son direct typiquement à des angles de +/- 30 degrés par rapport à la source : ce modèle, fondé sur une estimation statistique, permet de faire l'économie d'un réel calcul de sources images qui par ailleurs serait impossible sans représentation géométrique de la scène. Par ailleurs le fait de rapprocher spatialement les réflexions précoces du son direct permet d'insister perceptivement sur le facteur « présence » de la source. Au contraire, en augmentant cet angle, il est possible de jouer sur la largeur apparente de la source, jusqu'à finalement, lorsque cet angle devient important, insister sur la contribution de la salle dans la réponse impulsionnelle. Ce type de traitement permet donc d'introduire un paramètre de contrôle supplémentaire qu'il est possible de manipuler consciemment et qui serait difficile de retrouver dans le cadre d'une modélisation purement physique.

Le Spatialisateur IRCAM a fait l'objet d'une recherche importante au niveau de son interface de contrôle : à partir des paramètres « bas-niveau » inexploitables par un utilisateur, une étude a permis de mettre en évidence un nombre raisonnable de paramètres « perceptifs » adaptés et qui font sens aux compisteurs. Nous reviendrons sur cette interface dans la section 2.2 de ce document, lors de l'étude des interfaces de contrôle de la spatialisation. Nous reviendrons également sur le Spatialisateur IRCAM au paragraphe 6.4 où nous décrivons notre expérience de pilotage de ce système à l'aide de notre prototype, MusicSpace.

1.3.10 Conclusion

La spatialisation dispose donc aujourd'hui d'un grand nombre de techniques et de systèmes pour la mettre en œuvre. Chaque solution possède ses avantages autant que ses inconvénients,

et le choix d'une de ces techniques par rapport à une autre est le résultat d'un compromis entre un ensemble de paramètres parmi lesquels on trouve :

- La ou les tâche(s) de spatialisation à effectuer : simulation d'effet de salle, simulation de localisation angulaire, simulation de la directivité de la source,…
- La nature du système d'écoute utilisé : casque, paire de haut-parleurs, systèmes multipoints, systèmes holophoniques…
- La situation d'écoute : concert et grand nombre d'auditeurs, systèmes multimédia individuels, public immobile ou en déplacement, …
- Coût de l'algorithme et puissance de calcul disponible

S'orienter vers une méthode plutôt qu'une autre repose donc en grande partie sur les conditions dans lesquelles la spatialisation doit être effectuée, et le choix est généralement facile à effectuer. Au niveau des performances et du résultat sonore, un comparatif de ces différentes méthodes de spatialisation a été réalisé par Jean-Marc Jot, Véronique Larcher et Jean-Marie Pernaux (voir [Jot & Al, 1999]). Cet article récapitule également les différentes situations d'utilisation relatives à ces procédés.

Par ailleurs, chacune de ces techniques apporte un ensemble de paramètres de contrôle qui lui est propre. En particulier, le nombre et la nature de ces paramètres dépend du degré d'acuité de la simulation de spatialisation, ainsi que évidement de l'approche par laquelle le contrôle est abordé, l'approche physique ou l'approche perceptive, ou encore, dans certains cas une approche mixte physique et perceptive. Il est impossible de prendre en considération dans notre projet l'ensemble des paramètres de chacunes de ces techniques : ils seraient soit trop spécifiques à une technique, soit simplement non significatifs pour l'utilisateur. Cependant il existe un certain nombre de ces paramètres que l'on retrouve de manière quasi-systématique au travers de toutes les techniques : les positions géométriques des sources de manière absolue, ou relatives à la position d'un auditeur. Ces positions s'expriment en distances ainsi qu'en angles permettant de caractériser le trajet direct source – auditeur, ainsi que les différentes orientations des sources pour les systèmes qui rendent compte de leur directivité. Ce sont sur ces paramètres que nous proposons de nous concentrer, en particulier au travers des modules d'interfaçage des systèmes de spatialisation qui, comme nous l'expliquons dans la section suivante, visent à uniformiser la manière dont ces paramètres sont adressés pour tous les moteurs de spatialisation.

1.4 L'interfaçage des systèmes de spatialisation

Tel que nous l'avons montré, la spatialisation a au cours des dernières décades suscité l'intérêt d'un grand nombre de chercheurs, et fait l'objet de beaucoup d'études tant au niveau de la perception des sources sonores dans l'espace, qu'au niveau de la simulation de cette impression d'espace au moyen de procédés audionumériques. Techniques, algorithmes et mises en œuvre se sont multipliés, chacun abordant le problème sous un angle original, et dévoilant en conséquence une portée de paramètres de contrôle propre, spécifique à la méthode de spatialisation employée.

Aujourd'hui cette tendance à la prolifération s'inverse : devant la diversité des systèmes de spatialisation et la multitude de paramètres de contrôle qui les accompagne, certains efforts

d'uniformisation surgissent et visent à proposer une méthode générique de contrôle de ces systèmes, indépendamment de la technique sous-jacente employée. Une première explication a rapport à la portabilité des systèmes : en séparant des logiciels multimédia les parties bas niveau fortement liées au matériel utilisé, les concepteurs d'application favorisent les facilités de portage de leur travail vers d'autres plateformes matérielles et logicielles.

Ainsi, nous voyons apparaître des systèmes servant d'interface entre les composant logiciels utilisant la spatialisation et les systèmes de spatialisation eux-mêmes. Les logiciels musicaux pilotent alors la spatialisation des sources sonores qu'ils produisent dans un langage universel indépendant de la technique utilisée : l'intérêt fondamental de ce degré d'indirection supplémentaire est la possibilité de changer de manière directe le spatialisateur utilisé, lorsque en particulier la situation d'écoute change, et qu'une méthode de spatialisation devient plus adaptée qu'une autre (typiquement, comme nous l'avons montré, l'écoute individuelle au casque et la diffusion de concert sont des situations dans lesquelles des algorithmes de spatialisation sont préférables).

Des groupes de discussion tels que le «Interactive Audio Special Interest Group » [IA-Sig] ont précisément pour but de faciliter l'échange entre les concepteurs de logiciels, de matériel ou de contenu sonore. Ce groupe a déjà su influencer le développement de processus de standardisation dans le domaine de l'audio, ainsi que d'APIs (Application Programming Interface) par exemple pour l'environnement Windows de Microsoft. C'est également des discussions provenant de ce forum qu'ont émergé certains concepts importants d'OpenAL, que nous décrivons maintenant.

1.4.1 OpenAL

L'Open Audio Library (OpenAL) est le résultat d'une collaboration entre différents instituts et industriels, pour produire une API ouverte, indépendante des constructeurs, indépendante du système d'exploitation utilisé et orientée principalement vers la spatialisation du son. OpenAL est avant tout un moyen permettant de produire un résultat sonore spatialisé dans un environnement tridimensionnel : les techniques traditionnelles propres au mixage (effets de panoramiques gauche/droit par exemple) sont donc délaissées pour se tourner d'avantage vers des paramètres tels que la directivité des sources, l'atténuation due à la distance, ou l'effet doppler, ainsi que vers des notions liées aux environnements virtuels comme par exemple les réflexions, obstructions, transmission et réverbération, ainsi qu'il l'a été suggéré par les recommandations (rendering guidelines 3D Level 1 et Level 2) de IA-Sig (voir [MMA, 1998] et [MMA, 1999]).

Nous donnons ici à titre d'exemple une description des paramètres de base qu'il est possible de manipuler au sein de OpenAL, pour les objets « source » et « listener » (point d'écoute).

Paramètres communs aux sources et aux listeners :

Position	(x,y,z) un triplet de nombres flottants qui détermine la position de l'objet dans un repère à trois dimensions.
Vélocité	(vx, vy, vz) Il s'agit de la donnée de la vitesse instantanée de l'objet, mesurée elle aussi sur les trois axes, et nécessaire au calcul de l'effet Doppler par exemple.

Gain	(α) un facteur multiplicateur de l'amplitude compris entre 0 (atténuation totale) et 1, aucune atténuation. Une valeur de 0.5 équivaudrait à une atténuation de $20*\log_{10}(0.5) = -6dB$

Paramètres spécifiques aux listeners :

Orientation	$((at_x,at_y,at_z),(up_x,up_y,up_z))$ L'orientation d'un listener peut être totalement définie à partir d'une paire de vecteurs tridimensionnel, que l'on présuppose orthogonaux, et qui determinent la direction dans laquelle le listener est en train de regarder et la direction qui représente le « haut » pour le listener. Ce dernier vecteur permet par exemple, si le listener symbolise un être humain, de savoir si celui-ci penche la tête sur le coté.

Paramètres spécifiques aux sources :

Bouclage	Cette valeur logique détermine le comportement de la source lorsque le lecteur atteint la fin du bloc tampon. Si ce paramètre possède la valeur vrai, la lecture reprend automatiquement au début du bloc.
Mode relatif	Cette valeur logique détermine si les coordonnées dans le paramètre « position » sont à considérer de manière absolue dans le repère 3D, ou bien de manière relative par rapport au listener.
Orientation	(at_x,at_y,at_z) c'est un seul vecteur en trois dimensions, déterminant l'axe dans laquelle la source émet. L'orientation de la source est considérée comme moins fondamentale que celle des listener, et le fait de se limiter à un seul vecteur précise qu'il ne sera pas tenu compte de toute rotation de la source autour de son axe d'émission. Lorsque ce vecteur est de norme nulle (0,0,0), la source est considérée – par convention – comme omnidirectionnelle. Dans les autres cas, la source est considérée comme directionnelle et les deux paramètres de cônes suivants interviennent dans le calcul

Cône Intérieur & Cône Extérieur	Ces deux paramètres angulaires, exprimés en degrés, permettent une description simpliste de la façon dont la source rayonne. Typiquement, pour une source sonore vue par la source sous un angle α, un cône intérieur de valeur α_i et un cône extérieur de valeur α_e, le rayonnement peut être modélisé de la façon suivante : • si $\alpha < \alpha_i$: pas d'atténuation • si $\alpha > \alpha_e$: atténuation complète • si $\alpha_i < \alpha < \alpha_e$: l'atténuation varie avec une loi d'interpolation sur les angles.

1.4.2 DirectX – DirectSound3D

DirectX ([DirectX]) correspond à un ensemble de techniques destinées à rendre le système d'exploitation Windows de Microsoft le plus performant possible pour accueillir des applications riches en éléments multimédia, vidéo, animations tridimensionnelles et, en ce qui nous intéresse, rendu sonore tridimensionnel. L'API proposée offre au programmeur ou au concepteur d'applications multimédia, la possibilité de décrire simplement une scène sonore virtuelle, au moyen de sources et de points d'écoute et se charge d'en produire le rendu sonore soit au moyen de calculs internes soit en faisant appel aux capacités du périphérique sonore utilisé. Les paramètres de contrôle de la scène sont ceux de la description géométrique de la scène. L'ensemble de ces paramètres est extrêmement similaire à ceux proposés dans le système OpenAL, présenté précédemment.

Les sources ainsi que l'auditeur sont définis par des coordonnées géométriques de position et d'orientation dans un système de coordonnées à trois dimensions. Par ailleurs, les sources sonores possèdent une distance maximale (au delà de laquelle elles ne sont plus perçues) et une distance minimale en deçà de laquelle leur niveau sonore ne varie plus et reste figé à une valeur maximale. Ces caractéristiques permettent par exemple d'épargner le processeur lorsqu'une source trop distante du point d'écoute n'est plus perçue, et d'autre part de contourner la loi d'atténuation en $1/r^2$ selon laquelle l'intensité d'une source tendrait vers l'infini à mesure qu'elle s'approche de l'auditeur. Par ailleurs à chaque source est associé un ensemble de paramètres de directivité. Il s'agit de la définition de deux cônes imbriqués et centrés sur la source, déterminant trois zones, la zone intérieure aux deux cônes dans laquelle on considère que la source rayonne, la zone extérieure dans laquelle on considère que la source ne rayonne pas, et la zone intermédiaire dans laquelle est calculé un volume de transition (voir Figure 5). En théorie la simulation de la directivité d'une source peut être un calcul relativement complexe et coûteux lorsqu'on souhaite la restituer avec exactitude. Lorsque l'effet de salle est pris en compte par exemple, l'effet d'une source non orienté vers le point d'écoute peut être simulé en privilégiant le niveau des premières réflexions par rapport à celui du son direct de la source. En pratique, du fait que directX ne gère pas l'effet de salle, une approximation très élémentaire est proposée et ressemble à un simple étouffement du son (un filtrage passe-bas) lorsque la source « tourne le dos » à l'auditeur.

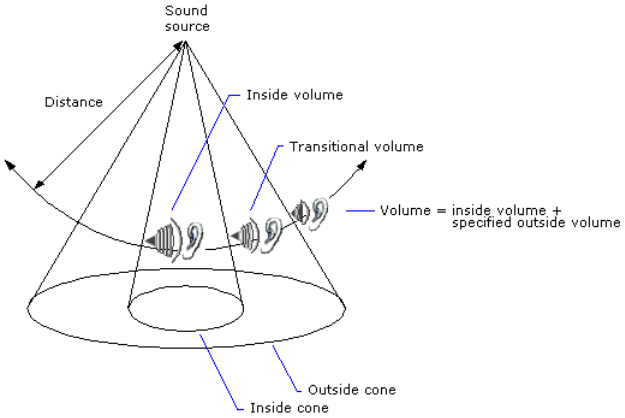


Figure 5 : représentation de la directivité des sources sous forme de cônes dans le système DirectX

En réalité, DirectX remplit au niveau sonore deux fonctions : d'une part il s'agit d'un Spatialisateur logiciel capable de transformer le signal monophonique d'une source sonore, en un signal multicanal (stéréo ou quadriphonique suivant le système de restitution choisi) spatialisé. Par ailleurs, lorsqu'un dispositif matériel sonore performant – une carte son « accélératrice » - est connectée à l'ordinateur, DirectX sait confier à celle-ci les tâches dont elle est capable, soulageant ainsi le processeur des calculs de traitement du signal et autorisant éventuellement un rendu plus performant.

DirectX permet donc de composer des scènes sonores spatialisées sans se soucier du matériel utilisé. En particulier le concepteur de ces scènes ne possède aucun moyen *a priori* de savoir quelle technique va être mise en œuvre pour simuler la spatialisation : c'est au dernier moment, celui de la restitution sonore, que ce choix est fait, en fonction du dispositif de restitution dont l'utilisateur dispose (capacité de calcul de sa carte son, et type de restitution pour casque, haut-parleurs stéréo ou quadraphonie).

1.4.3 MPEG-4

MPEG-4 (de Motion Pictures Experts Group, layer 4) est sans aucun doute l'effort de normalisation de la description de scènes multimédia le plus conséquent à l'heure actuelle ([Koenen, 1999]). L'objectif de ce standard est de spécifier clairement et de la manière la plus générique possible un ensemble de descripteurs et un langage correspondant approprié, introduisant suffisamment de flexibilité pour définir a priori la majeure partie des scènes multimédia interactives.

La norme se partage en un ensemble de composants Graphiques (2D et 3D), Audio (2D et 3D également) et « Système » permettant différents types de rendus, ainsi que les mécanismes nécessaires à la gestion temporelle des événements et la prise en compte des interactions avec l'utilisateur. La scène est entièrement bâtie autour d'une structure « objet » qui permet non seulement de la manipuler facilement, mais aussi d'en construire une représentation personnalisée au niveau de chaque terminal de visualisation MPEG-4.

Dans le cas de rendu sonore 3D, la scène sonore est décrite au travers de la notion de source sonore à partir d'un ensemble de deux jeux différents de paramètres, les paramètres « physiques » et les paramètres « perceptifs ». Les paramètres physiques tentent d'évaluer la qualité de chaque trajet acoustique source – listener à partir de la donnée géométrique de

l'environnement (position des sources, description des parois,…) et des qualités acoustiques de chaque composant (coefficient de réflexion des parois, par exemple). L'approche perceptive, au contraire, ignore la représentation physique de l'environnement et vise à la définition de la qualité acoustique à partir d'un ensemble de critères liés à la perception, en des termes que le créateur de contenu saura manipuler. L'une et l'autre représentation a ses propres avantages et inconvénients et il appartient au concepteur de la scène de décider laquelle lui semble la mieux adaptée au contenu qu'il souhaite créer. Cependant les deux représentations on en commun un ensemble de paramètres liés à la position des sources sonores de manière absolue ou relative à une position d'écoute : ce sont à ces paramètres que nous nous proposons de nous intéresser dans notre projet.

Le standard MPEG-4 permet donc la description de scènes sonores de manière souple, soit par le biais d'une approche « physique » de l'environnement, soit par une approche « perceptive ». Ces descriptions présentent de plus l'avantage d'être totalement indépendantes du moteur de spatialisation utilisé et en particulier du système de restitution (casques, haut-parleurs stéréophoniques,...) : un décodeur compatible MPEG-4 saura donc interpréter les différents types de description et rendre au mieux la scène sonore en fonction du système de diffusion souhaité.

Ces considérations sont capitales au sein de notre étude : elle permettent de nous affranchir d'un certain nombre d'aspects techniques relatifs aux techniques de spatialisation elles-mêmes pour nous concentrer directement sur le contrôle de ces systèmes au travers de ce niveau d'indirection ou d'interfaçage que nous venons de présenter. Nous étudions maintenant, dans les sections suivantes de ce document, les interfaces proprement dites de contrôle de la spatialisation.

Chapitre 2
Interfaces de Contrôle de la spatialisation du son.

Les recherches sur le sujet précis des interfaces de contrôle de la spatialisation sont relativement peu nombreuses. Beaucoup moins nombreuses, pour sûr, que les recherches menées en traitement du signal sur les moteurs de spatialisation eux-mêmes.

Pour les interfaces utilisateurs, souvent, les travaux s'intéressent au problème du contrôle de paramètres de manière générale et ne proposent pour la spatialisation de sources sonores aucune modalités de contrôle spécifiquement liées au domaine mais uniquement une application parmi d'autres de leur prototype.

Nous faisons état, dans ce chapitre, de certaines études qui se sont penchées particulièrement sur le contrôle de la spatialisation, en essayant de mettre en évidence différentes problématiques ou différents manières d'aborder le contrôle de la spatialisation. Nous commençons avant tout par une étude et description inévitable de la console de mixage, précurseur de tous les systèmes de spatialisation, qui règne encore sur son domaine.

2.1 Les consoles de mixage

Les consoles de mixage sont apparues au moment où il est devenu souhaitable ou nécessaire d'ajuster les paramètres de reproduction de plusieurs sources sonores simultanément, soit dans le cadre d'une diffusion de concert, soit dans l'objectif d'un enregistrement. L'effet recherché vise non seulement à ajuster les sources de manière individuelle, mais aussi à atteindre un équilibre sonore soit en timbre, soit en intensité, entre celles-ci.

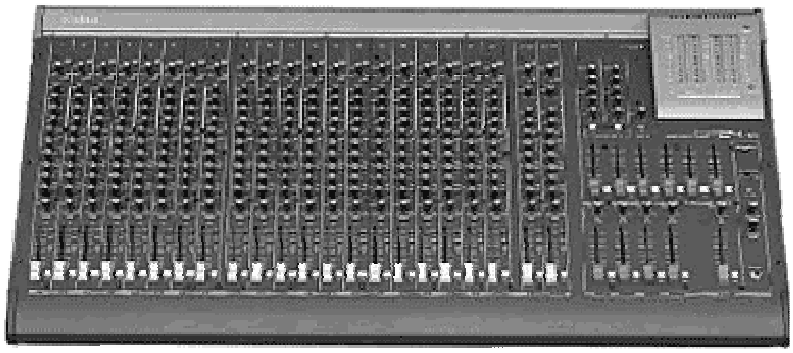


Figure 6 : Vue d'une console de mixage traditionnelle

Les possibilités d'utilisation de la console de mixage sont multiples : dans le cadre d'une diffusion de concert, par exemple, les « sources » correspondent à des microphones positionnés au sein d'un ensemble instrumental moderne pour lequel – contrairement à un orchestre symphonique par exemple – les rapports de force entre les différents pupitres sont tels, qu'il est nécessaire d'avoir recours à des procédés artificiels pour atteindre un équilibre sonore. Ces microphones peuvent être également dirigés vers un instrument unique, complexe – comme le piano par exemple – lorsqu'on souhaite en capter les sonorités avec précision. La console de mixage sert alors à doser la contribution de chacun de ces microphones et à rassembler des détails sonores captés sous différents angles de l'instrument de sorte que leur mélange produise une image la plus fidèle et la plus réaliste possible.

La console de mixage est une chaîne matérielle de traitement de signal que l'on réplique à l'identique dans une même « boite » autant de fois qu'il est de microphones ou de sources de signal à traiter. Les capacités d'une console se mesurent donc avant tout en nombre de pistes disponibles simultanément et se décrivent sous la forme d'un triplet par exemple de « 16/8/2 » signifiant 16 pistes d'entrée, 8 bus auxiliaires, et deux canaux de sortie. Schématiquement, les 16 pistes d'entrée sont ajustées en niveau, en dynamique et en timbre, avant d'être renvoyées soit vers les bus auxiliaires par exemple, dans le cas d'un enregistrement multipistes, soit vers les canaux de sortie pour une diffusion stéréophonique.

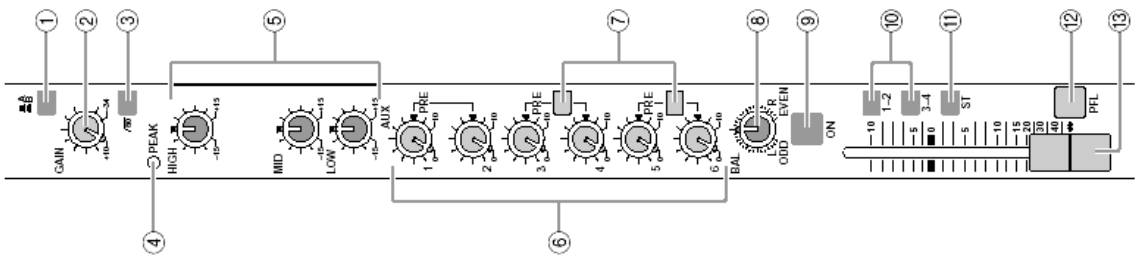


Figure 7 : représentation physique d'une « tranche » d'une console de mixage Yamaha (pivotée de 90° vers la gauche)

L'interface de contrôle, matérielle (et donc figée), est directement liée aux ressources physiques de traitement du signal correspondantes : pour preuve, chacun des contrôles ou potentiomètres présentés à l'utilisateur sur la face avant de la console (Figure 7) se retrouve directement dans la représentation sous forme de « block diagramme » de la console (Figure 8), représentation simplifiée du circuit électronique correspondant.

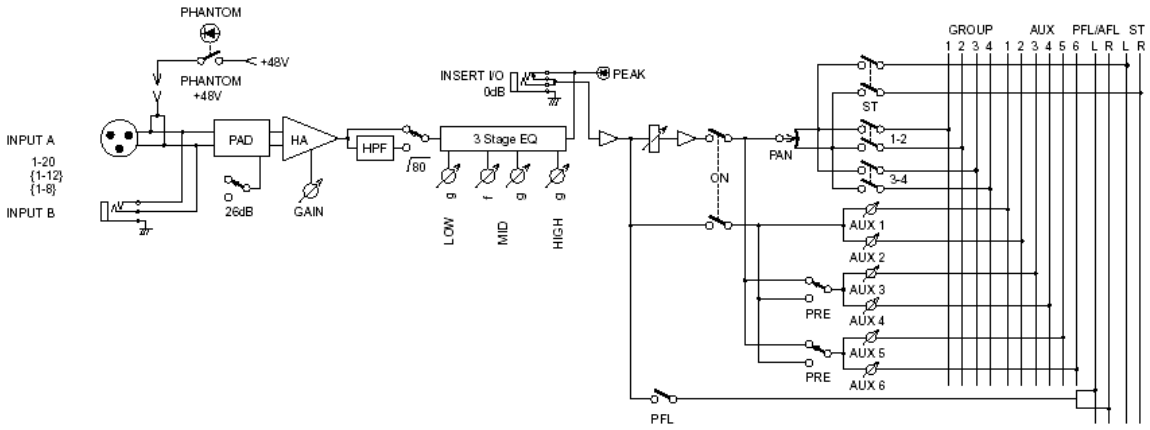


Figure 8 : représentation sous forme « block diagramme » d'une partie d'une console de mixage

C'est donc une interface de contrôle très bas niveau, au travers de laquelle l'utilisateur ajuste individuellement des paramètres directement issus de la chaîne de traitement du signal sonore sous-jacente. En revanche la répartition physique des « boutons » sur l'interface n'est pas laissée au hasard et un effort d'ergonomie est fait à ce niveau : en particulier, les boutons les plus souvent utilisés sont privilégiés par un accès plus facile (plus proches de l'utilisateur), ainsi que par une taille plus importante pour permettre une plus grande marge d'action et donc une plus grande précision. Typiquement, le contrôle de gain en amont de la chaîne de traitement se trouve généralement tout en haut de la tranche d'entrée, et de taille relativement petite. Ce contrôle sert à amplifier ou limiter le signal entrant pour l'amener à une échelle « raisonnable », en particulier en regard des autres signaux d'entrée. Ce paramètre est généralement ajusté une fois en début de session et n'est ensuite plus utilisé, laissant alors le contrôle au « curseur de niveau », le plus grand, et le plus proche de l'utilisateur.

Cette interface permettant de contrôler une piste est répétée, à l'identique, autant de fois qu'il y a de chaînes de traitement du signal correspondante pour former le « bloc d'entrée » de la console. On y trouve là peut être l'origine de ce principe général consistant à organiser le mixage autour de la notion de « piste sonore » au sens de la ressource matérielle et non au sens de la voix d'une polyphonie par exemple. Précisément un instrument complexe à capter pourra faire l'objet de l'utilisation de plusieurs pistes de la console de mixage. Inversement, une seule piste pourra être suffisante pour capter tout un ensemble d'instruments. Cette différence dans la notion de « piste » au sens de ressource de traitement et au sens musical sera davantage détaillée ultérieurement, dans la section concernant le « parsing » des fichiers MIDI située en annexe de ce document.

Par ailleurs, la représentation sous forme de « banc » de curseurs alignés horizontalement pour le contrôle de l'amplification individuelle de chaque entrée permet, lorsque les sources sonores sont toutes normalisées, de se donner une représentation visuelle des niveaux sonores relatifs à chaque source et peut aider l'ingénieur du son dans sa démarche d'écoute analytique.

La tâche consistant à effectuer un mixage correct, c'est-à-dire de sorte à mettre en avant un équilibre cohérent des instruments, est complexe. Elle nécessite un savoir faire particulier, objet du métier d'ingénieur du son. Celui-ci développe une « oreille », des capacités d'analyse et une maîtrise des instruments de mixage lui permettant, au vu de la scène sonore, de décider quel est la nature de l'équilibre sonore à atteindre, et comment y parvenir à partir des modalités de contrôle qui lui sont proposées. Ses choix sont influencés par la prise en considération, par exemple, du style musical abordé, et sont conditionnés par les possibilités d'intervention dont il dispose, tant au niveau des ressources en traitement du signal, que des possibilités de contrôle qui y sont reliées.

L'interface de contrôle du mixage présente sur les consoles est restée suffisante et parfaitement adaptée pour un ensemble d'utilisations bien déterminé, faisant intervenir des scènes sonores composées d'un nombre relativement « limité » d'instrumentistes et n'évoluant pas trop au cours d'une même session. Il s'agit typiquement, un mixage stéréophonique pour une formation musicale populaire. Le succès des concepts de contrôle de la console de mixage est tel que ceux-ci sont maintenant répliqués de manière quasi systématique dans la majorité des interfaces informatiques de contrôle du mixage, comme en témoigne par exemple la vue (Figure 9) de l'application « mixer » de contrôle de l'amplification des différentes entrées physiques ou logiques d'une carte sonore d'ordinateur. Une fois de plus, nous constatons que les contrôleurs associés au réglage du niveau sonore occupent sur l'interface une place nettement plus importante que ceux liés au réglage de panoramique, paramètre jugé moins important.

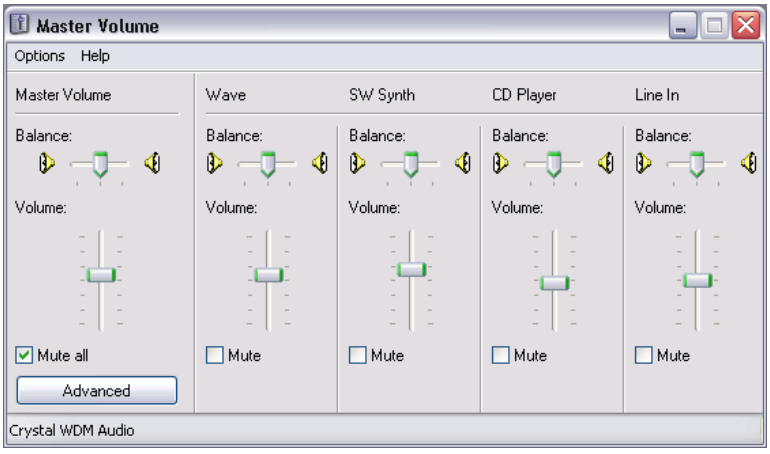


Figure 9 : reproduction informatique des concepts de la console de mixage

Dans les situations de mixages plus complexes, lorsque la précision nécessaire du résultat devient plus importante, ou lorsque le nombre de sources sonores à mixer simultanément augmente, l'interface de contrôle des consoles de mixage traditionnelles n'est plus suffisante.

Figure 10 : vue du logiciel VDesk, système d'automation de la console Yamaha O2R.

Une avancée majeure dans ce domaine a été le contrôle numérique des consoles permettant de piloter certains de leurs paramètres depuis un outil informatique. Initialement, c'est le paramètre « mute » permettant de couper la sortie de chacune des tranches qu'il a, le premier, été possible de contrôler dynamiquement depuis un logiciel de séquencement MIDI. Par la suite, l'ensemble des paramètres a été ouvert au contrôle externe, en même temps que sont apparus les systèmes d'automation (tel que le logiciel VDesk, pour Macintosh, développé par la société JLCooper Electronics, et capable d'automatiser une console Yamaha O2R, représenté en Figure 10 [VDesk]) permettant d'effectuer un mixage en plusieurs étapes, en mémorisant à chaque étape l'ensemble des valeurs des paramètres, et en les raffinant progressivement tour à tour. Par ailleurs, la commande numérique a également permis l'introduction de « mémoires » permettant mémoriser l'ensemble des valeurs de paramètres et ainsi de passer instantanément d'une mémoire de scène à une autre.

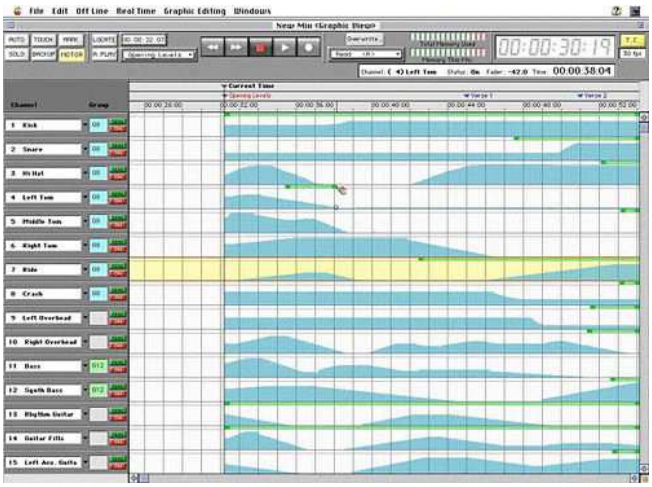


Figure 11 : vue du logiciel VDesk (2), chaque paramètre apparaît comme un piste d'enregistrement

Pour finir, si les paramètres du mixage des consoles actuelles restent de manière générale très bas niveau, nous mentionnons tout de même deux caractéristiques des consoles de mixage qui nous intéressent tout particulièrement puisqu'elles permettent, dans une certaine mesure,

d'établir des relations entre les paramètres de mixage de pistes différentes. Il s'agit en premier lieu de l'utilisation des pistes dites « sous-groupes » dont le rôle initial est de réduire le nombre de canaux à mixer simultanément de manière à être compatible, par exemple, avec le nombre de canaux qu'un magnétophone multipiste est capable d'enregistrer simultanément (typiquement 8 ou 16 dans la majorité des cas, seuls les plus grands studios d'enregistrement sont dotés de machines permettant d'enregistrer jusque 48 pistes simultanément). Ces canaux représentent donc une étape intermédiaire du mixage sur laquelle il est possible d'agir sans défaire l'équilibre construit entre les pistes d'entrée qui contribuent à ce sous-groupe.

La Figure 12 représente schématiquement une partie d'une console de mixage et donne un exemple concret d'utilisation de ces sous-groupes : dans cet exemple, cinq microphones placés à diverses position autour d'une batterie alimentent les pistes d'entrées de la console, marquées GC (grosse caisse), CC (caisse claire), CY1 (cymbale 1), CY2 (cymbale 2) et HH (hi-hat). Ces pistes sont ajustées individuellement en niveau et en panoramique puis affectées aux sous-groupes 1 et 2 de la console. L'affectation aux sous-groupes va généralement par paire (ex sous-groupe 1 et 2) et la quantité de signal envoyée à chacun dépend de la valeur de panoramique de la piste d'entrée (typiquement lorsque le panoramique est tourné vers la gauche, le signal est envoyé aux sous-groupes impairs, et lorsque le panoramique est tourné vers la droite, le signal est dirigé vers les sous-groupes pairs, une valeur intermédiaire de panoramique effectuant une pondération correspondante sur la quantité de signal envoyée aux sous-groupes).

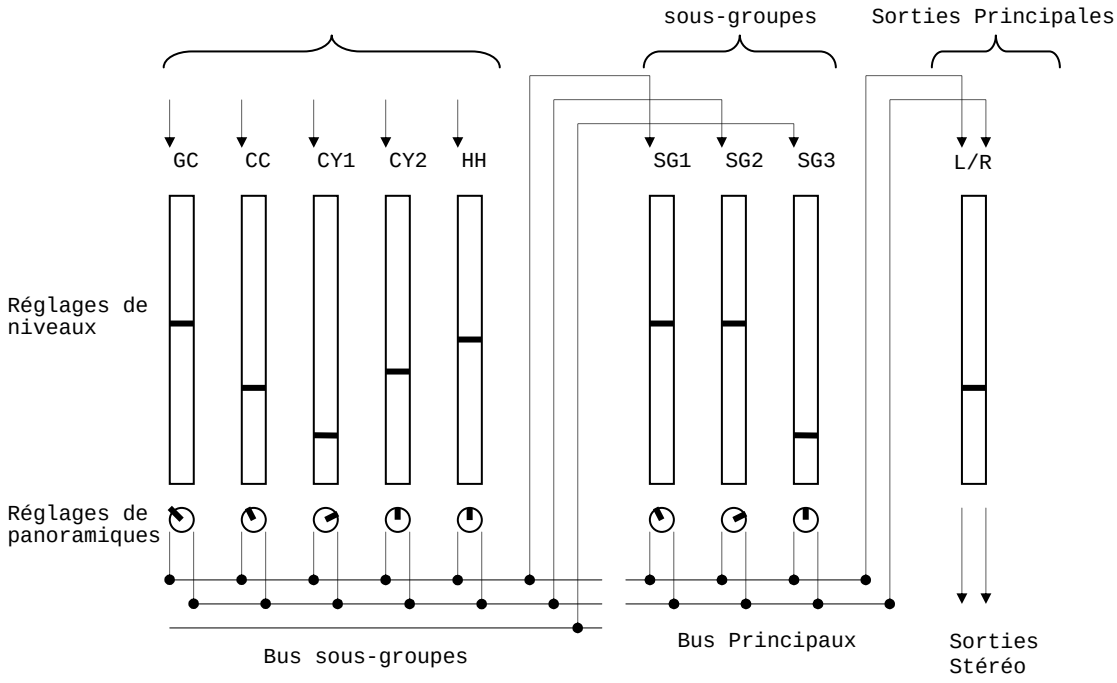


Figure 12 : réduction du nombre de pistes d'entrée à l'aide des pistes de sous-groupes

Au niveau du sous-groupe, il est encore possible d'intervenir sur le réglage de niveau. Une telle manipulation est équivalente au fait de modifier simultanément – et d'une quantité équivalente – tous les curseurs de réglage de niveaux des pistes d'entrée contribuant à ce sous-groupe. De même, l'action sur les réglages de panoramique au niveau des sous-groupes redistribue les

pistes d'entrée dans l'image stéréophonique, mais conserve l'équilibre conçu initialement indépendamment pour chacune des sources.

En utilisant les sous-groupes, l'ingénieur du son peut donc construire un ensemble de contrôles intermédiaires du mixage, plus haut niveau que le contrôle direct des pistes d'entrée, lui permettant d'équilibrer des parties du mixages sans altérer l'équilibre qu'il a conçu au sein de ces parties. Ce type de manipulations lui garantit une plus grande précision et dextérité en situations de diffusion de concert, ou d'enregistrement.

Par ailleurs, certaines consoles modernes permettent également d'établir un lien symbolique entre deux canaux d'entrée « voisins » par l'intermédiaire de la notion de « paire stéréo ». Il est fréquent que les canaux d'entrée d'une console fonctionnent par paires, soit parce qu'il s'agit de signal provenant d'une machine stéréophonique, soit par ce que ces pistes d'entrée correspondent à un couple de microphones coïncidant. Dans ce cas, il est logique de penser que les modifications apportées à une piste de la console devraient l'être également à la piste en correspondance. L'action de relier deux pistes sous forme de paire fige les écarts de réglages de ces deux pistes en niveau autant qu'en panoramique. L'ingénieur du son peut alors par exemple déplacer aisément la position de l'image spatiale de cette paire dans la scène sonore sans en modifier la largeur. Nous montrerons (voir section 6.3.1) comment il est possible de retrouver ou reconstruire ces fonctionnalités à l'aide de contraintes au sein de MusicSpace.

2.2 L'interface perceptive du Spat

Le Spatialisateur de l'IRCAM a fait l'objet d'une recherche approfondie sur les modalités de contrôle par un utilisateur (voir [Jullien, 1995]). Ces études, de nature psycho perceptive, ont permis de réduire l'ensemble des paramètres initiaux – trop nombreux (de l'ordre de la centaine par source) et trop peu signifiants – en un sous ensemble de taille acceptable (une dizaine) et directement compréhensibles pour l'utilisateur.

L'interface de contrôle du spatialisateur IRCAM permet donc de décrire suivant un ensemble de paramètres « perceptifs » la qualité acoustique du trajet qui sépare la source sonore de l'auditeur. Ces paramètres ont été adoptés et intégrés à la norme MPEG-4 comme une manière possible de décrire une scène sonore. La Figure 13 donne un aperçu de l'interface SpatOper de contrôle du Spat. Cette interface réalisée dans le logiciel MAX ([Puckette, 1988]) représente les différents paramètres de contrôle à la fois sous forme de curseur ajustable (a la manière d'une console de mixage, donc) et sous forme de boite valeur donnant sous forme numérique la grandeur représentée par la valeur du curseur. Elle reprend donc le principe d'alignement de curseurs énoncé par la console de mixage, mais s'en distingue en y attachant non plus un paramètre de contrôle élémentaire de la chaîne de traitement du signal, mais une grandeur qualitative dont la valeur influence un ensemble de paramètres élémentaires. La taille visuelle de ces curseurs est en relation directe avec l'importance du paramètre correspondant, telle qu'elle a pu être évaluée lors de l'étude (approximativement le pourcentage de l'inertie expliquée dans l'analyse en composantes principales). Ainsi, les paramètres de présence de la source autant que de la salle, ainsi que le temps de réverbération tardive, jugés important, disposent d'un curseur de taille plus importante que les autres.

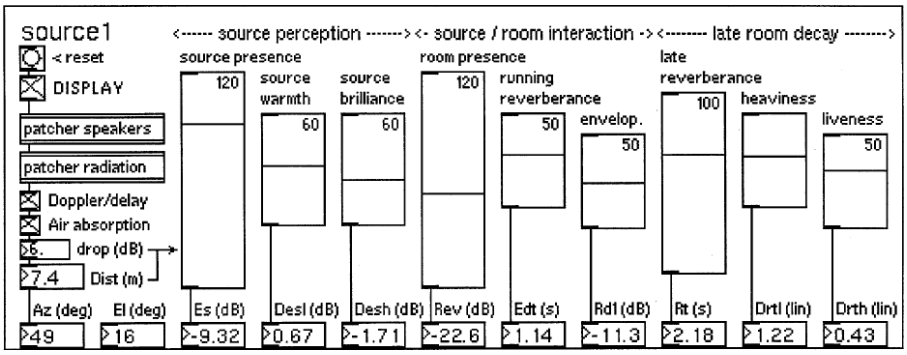


Figure 13 : « SpatOper » : l'interface perceptive du Spatialisateur IRCAM

Le Spatialisateur de l'IRCAM est capable de générer un effet de salle en se passant complètement de la description physique de celle-ci, juste à partir de sa description perceptive. Cependant un ensemble de paramètres « physiques » subsiste tout de même : il s'agit essentiellement de la distance à la source ainsi que de la manière dont elle est vue par rapport au point d'écoute, tels que la distance, l'azimut, l'élévation, le « pitch », le « yaw » et le « rolloff ». D'autres paramètres physiques permettent également de décrire la manière dont la source rayonne.

Malheureusement, ces paramètres « physiques » sont difficilement ajustables depuis le SpatOper. Le réglage de l'azimut, par exemple, se fait au moyen d'une zone de saisie de texte (voir Figure 14, boite de texte en bas à gauche) sous forme de nombre en degré : ceci est d'autant contestable qu'il s'agit d'un des paramètres que l'on est amené à modifier le plus fréquemment au cours d'une session. L'interface « CIRCLE » complète donc l'interface SpatOper de contrôle du Spatialisateur IRCAM précédemment décrite en proposant une approche dans laquelle priorité est donnée à ces paramètres physiques.

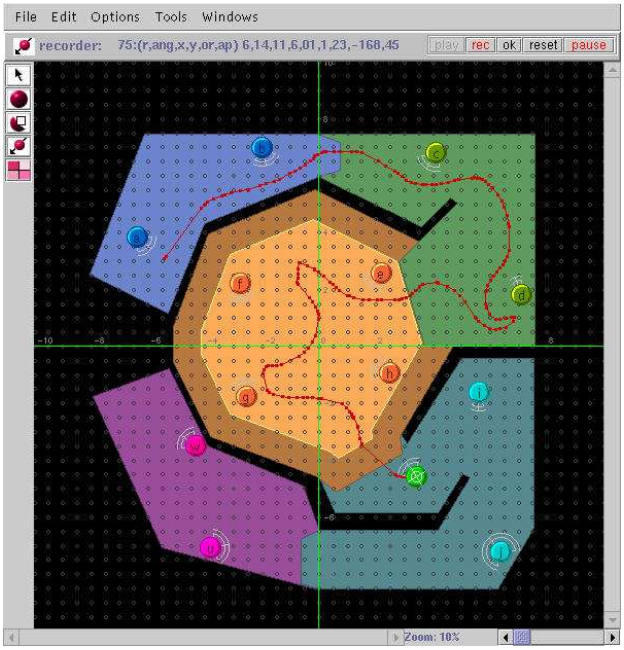


Figure 14 : Interface « Circle » (1) du Spatialisateur IRCAM

Cette interface représente les différentes sources sonores dans un espace à deux dimensions sous la forme d'une projection plane, vue de dessus. Par ailleurs, des zones colorées, aux contours délimités par des polygones représentent « l'occupation au sol » de salles, possédant chacune ses propriétés acoustiques. L'utilisateur se représente ainsi d'un seul coup d'œil l'ensemble des distances relatives des sources sonores, ainsi que leurs paramètres d'azimut, leur orientation et paramètres de rayonnement, par rapport à la position de l'auditeur d'une part, mais aussi, d'autre part, par rapport aux autres sources sonores présentes dans la scène.

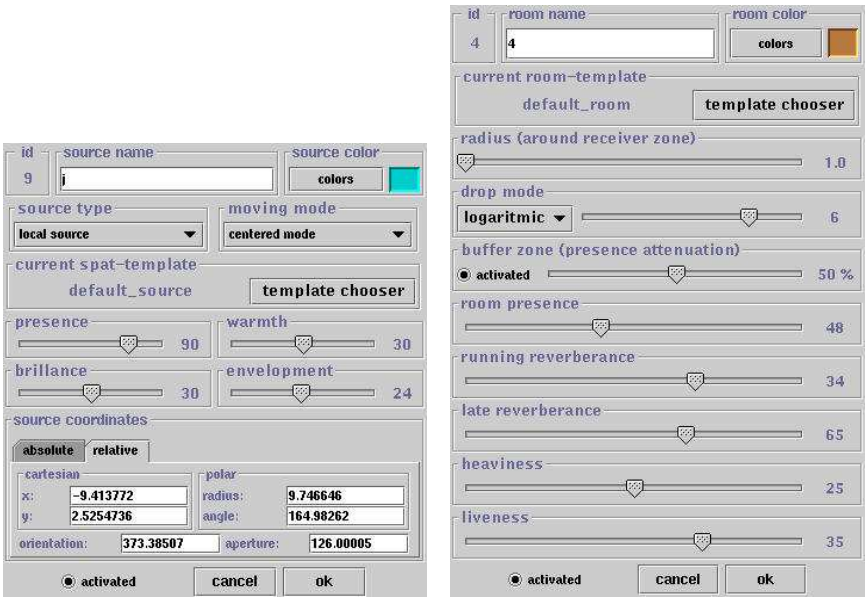


Figure 15 : Interface « Circle » (2) du Spatialisateur IRCAM.
Représentation des paramètres liés à la source (gauche) et à la salle (droite).

Les paramètres perceptifs du Spatialisateur IRCAM se retrouvent alors dans deux fenêtres de dialogue distinctes (Figure 15) auxquelles on accède soit par l'intermédiaire des objets sources (à gauche sur la Figure 15 qui reprend l'ensemble des paramètres perceptifs relatifs à la source), soit par l'intermédiaire des polygones décrivant graphiquement les « salles », pour les paramètres qui y sont relatifs (à droite sur la Figure 15). Un choix arbitraire est fait ici de relier les paramètres relatifs à la salle au polygone plutôt qu'au point d'écoute de sorte que plusieurs points d'écoute situés dans un même polygone partagent les mêmes valeurs de paramètres. La situation opposée, dans laquelle les propriétés acoustiques de la salle seraient liées non plus au polygone mais à l'objet récepteur, pourrait être tout aussi envisageable, dans le cadre d'applications à la téléconférence par exemple.

2.3 Holophon

Holophon est un projet développé au GMEM (Centre National de Création Musicale) destiné à produire des outils informatiques permettant la spatialisation du son [Pottier, 2000] et [Cabaud & Pottier, 2002]. Ce projet se compose de deux composants logiciels : Holo-Spat, le moteur temps réel de spatialisation, écrit dans l'environnement Max/MSP et Holo-Edit, un éditeur graphique permettant de décrire la spatialisation sonore sous la forme d'une ensemble de trajectoires qui seront ensuite exploitées par le spatialisateur.

La tâche délicate qui consiste à décrire des trajectoires – même dans un espace bidimensionnel – impose l'utilisation simultanée de plusieurs systèmes de représentation. Ainsi une vue principale de l'application (voir Figure 16) représente la scène sous la forme d'une vue d'oiseau, représentant au mieux le plan horizontal privilégié dans le cadre de la spatialisation du son. Dans cet espace, bordé par la représentation des haut-parleurs, l'utilisateur a la possibilité d'observer et de manipuler manuellement les trajectoires qu'il a construites.

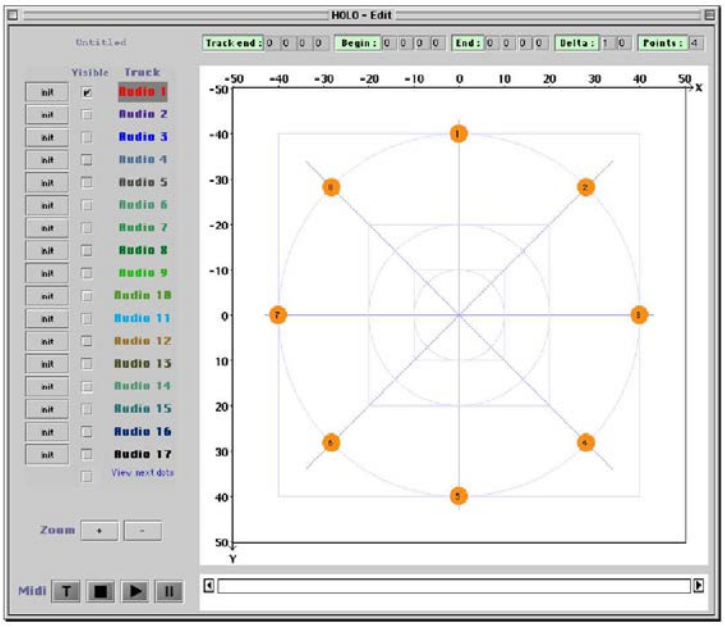


Figure 16 : vue de l'interface « Holo-Edit » (1), fenêtre principale

Par ailleurs, un éditeur temporel permet de visualiser la trajectoire en privilégiant l'axe du temps. Les trajectoires sont alors représentées sous la forme d'un ensemble de courbes, représentant chacune des coordonnées en fonction du temps. Ce type de représentation est nécessaire pour visualiser et éventuellement éditer certains paramètres importants relatifs au temps. En particulier, il permet de mettre clairement en évidence l'effet d'accélération de la trajectoire rouge sur l'exemple proposé en Figure 17.

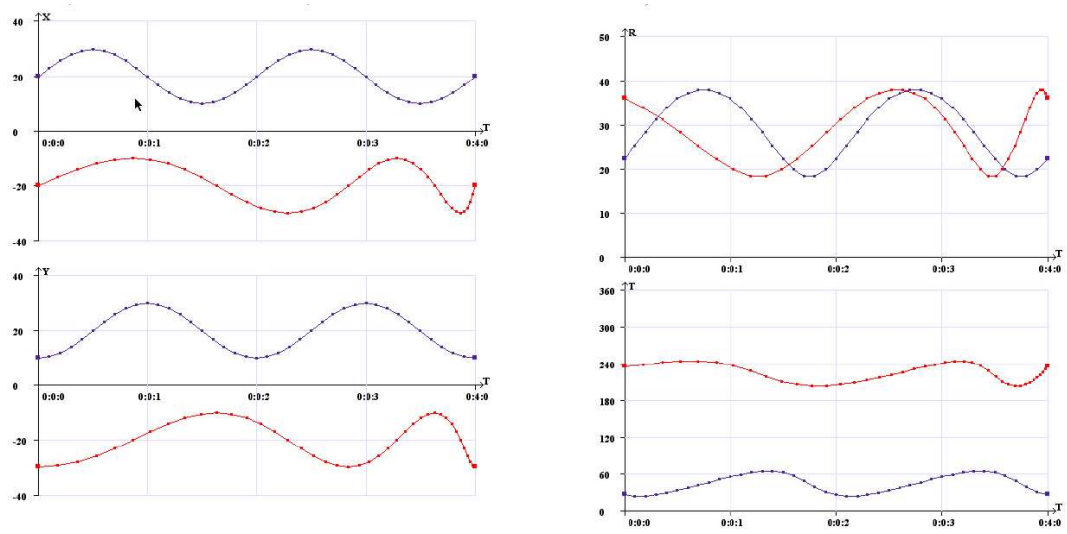


Figure 17 : vue de l'interface « Holo-Edit » (2), l'éditeur temporel

Mais ce qui fait l'originalité du logiciel Holo-Edit est sans aucun doute la possibilité de définir ces trajectoires de façon algorithmique. Un ensemble de primitives de construction est mis à disposition (cercles, spirales, trajets aléatoires…), ainsi qu'une base de fonctions de transformations élémentaires (symétrie, rotations, translations,…). L'utilisateur construit donc les trajectoires qu'il souhaite, en combinant ces fonctions de bases, par l'intermédiaire d'un script.

Les préoccupations du GMEM ont toujours été proches des objectifs de créations. Holophon a donc toujours cherché à répondre à une demande des utilisateurs : il été utilisé au sein de la réalisation de diverses pièces comme par exemple « Resonant Sound Spaces » de Jean-Claude Risset [Risset & Al, 2002].

2.4 MoveInSpace

Dans le même esprit que pour le système Holophon présenté précédemment, les recherches menées par Todor Todoroff, Caroline Traube et Jean-Marc Ledent au Laboratoire d'informatique Musicale de la Faculté Polytechnique de Mons s'articulent autour des interfaces de contrôle pour la commande d'instruments virtuels dans un premier temps [Todoroff & Traube, 1996] et plus particulièrement d'instruments de spatialisation par la suite [Todoroff & Al, 1997].

Leur projet, « MoveInSpace » est une interface de contrôle fonctionnant sous le système d'exploitation NeXTSTEP, et permettant de construire et visualiser des trajectoires dans un espace tridimensionnel à partir de différentes représentations. Ce système est connecté – sous la forme d'une communication de type client-serveur – à un moteur de traitement du son temps réel (FTS en l'occurrence [Viara & Puckette, 1990]) effectuant la spatialisation de signaux sonores.

Pour pallier la difficulté spécifique que ce projet aborde, la construction de trajectoires en 3 dimensions, MoveInSpace montre la scène sous différentes représentations. Une première vue sous forme de perspective permet de se représenter la scène mais n'est pas particulièrement

adaptée aux manipulations d'un utilisateur. D'autres formes de représentation, telles qu'une projection en deux dimensions et un éditeur temporel sont proposées : elles ne donnent qu'une vue incomplète de la scène mais se prêtent bien aux manipulations de l'utilisateur.

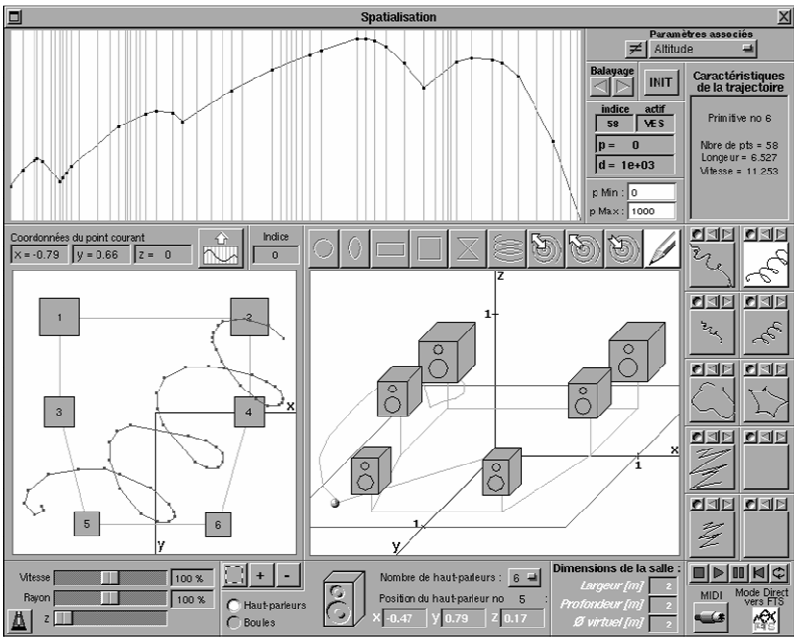


Figure 18 : MoveInSpace (1), L'éditeur de trajectoires

Par ailleurs en dehors de la commande habituelle par la souris les auteurs s'intéressent également à d'autres modalités de contrôle par l'intermédiaire de diverses interfaces incluant en particulier un « DataGlove », gant muni de quatre détecteurs de courbure associés aux phalanges d'une part, et d'un système de positionnement à ultrasons. Un tel dispositif permet, d'une seule main, d'exprimer conjointement une position dans l'espace ainsi que quatre valeurs de paramètres qui y seraient reliées.

Le système a été testé avec succès avec divers moteurs de spatialisation : un système interne, construit dans FTS, mais aussi des système externes tels qu'un acousmonium contrôlable par VCA, le spatialisateur de l'IRCAM ou encore une console de mixage de type Yamaha O3D.

Finalement, MoveInSpace présente un intérêt tout particulier du fait qu'il s'agit d'un système complètement ouvert qu'il a été possible d'intégrer au sein d'un environnement plus général : « The Interfaces Manager ». Ce dernier est capable d'analyser quels sont les paramètres d'entrée et les paramètres de sortie d'une application et propose donc un environnement (conceptuellement assez proche du gestionnaire d'applications de midishare : « MSConnect » [Orlarey & Lequay, 1989]), permettant de relier individuellement les paramètres de sortie d'une application aux paramètres d'entrée d'une autre application.

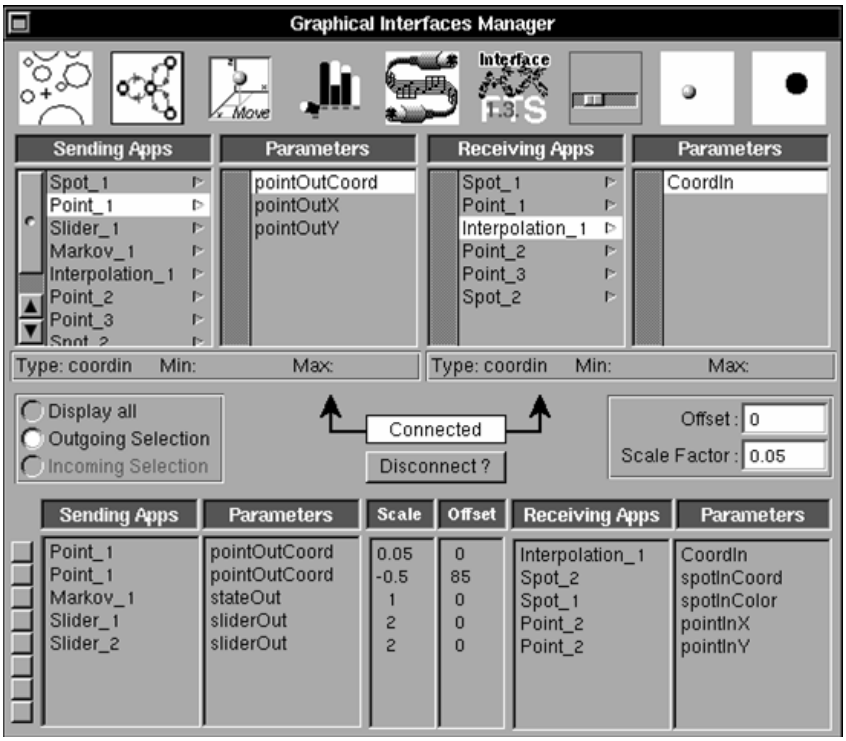


Figure 19 : MoveInSpace (2), Le gestionnaire d'applications

Il n'est donc pas ici question de passer par une interface intermédiaire de description de scène spécialement conçue pour la spatialisation comme vu précédemment dans la section 1.4 mais il est possible d'affecter un à un, les paramètres de sortie de MoveInSpace aux paramètres de contrôle d'un système de spatialisation et de mémoriser cette table de correspondances dans un fichier : ce choix rend la description d'un jeu entier de correspondances de paramètres certainement plus longue et fastidieuse, mais offre en contrepartie une plus grande souplesse d'application.

2.5 E.A.G.L.E.

L'univers des jeux vidéo, et de façon plus générale celui des réalités virtuelles pose un problème différent : les différentes modalités sensorielles (visuelle, auditive, et parfois proprioceptive) mises en jeu contribuent à l'établissement de la sensation d'immersion recherchée dans ce domaine d'application. Le contrôle de la spatialisation se fait donc directement au sein du logiciel, de façon inhérente à l'exploration de l'utilisateur, à partir de paramètres géométriques propres au monde visité.

Ainsi, ce domaine applicatif ne nécessite pas d'outils de contrôle de la spatialisation à proprement parler, mais requiert un important « travail auteur », en amont, au cours duquel le créateur relie les éléments géométriques du monde virtuel à un ensemble de descriptions de qualités acoustiques.

C'est précisément l'objectif du logiciel E.A.G.L.E. (Environmental Audio Graphic Librarian Editor) [EAGLE] : cet outil, conçu par la compagnie Creative Labs, permet d'exploiter pleinement les possibilités de traitement du signal « EAX » (Environmental Audio Effect) de leurs cartes sonores de type Sound Blaster. L'API « EAX » s'inscrit en complément des

paramètres de contrôle de DirectSound3D de DirectX pour y ajouter une description acoustique de l'effet de salle, augmentant l'impression de réalisme ainsi que la précision en matière de localisation des sources sonores – et en particulier en ce qui concerne la notion de distance – pour laquelle les premières réflexions (générées dans l'effet de salle, donc) jouent un rôle important.

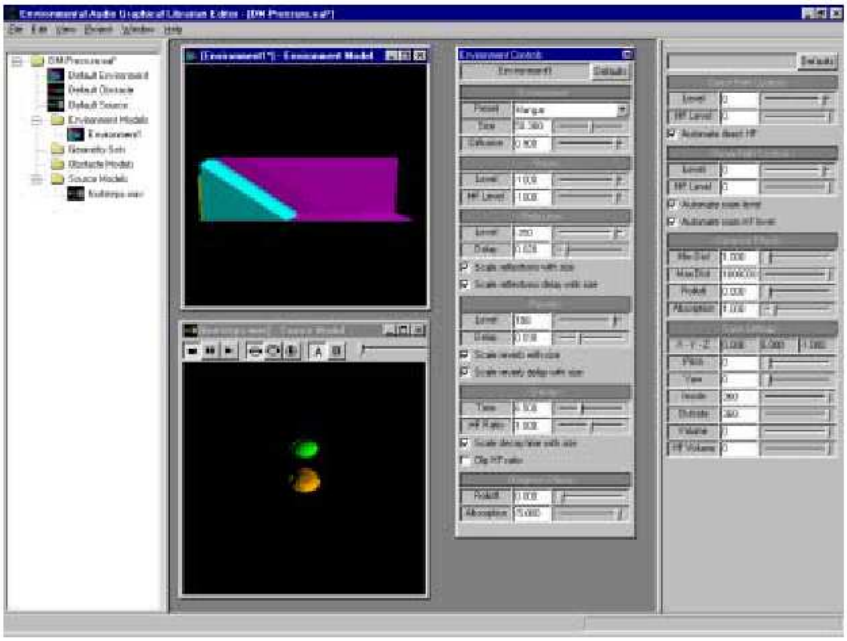


Figure 20 : vue du logiciel E.A.G.LE. (1) Fenêtre d'édition d'environnements

L'auteur décrit donc dans un premier temps une librairie d'effets environnementaux à partir d'un jeu de paramètres essentiellement physiques (temps de réverbération et courbes de décroissance). L'interface utilisateur (voir Figure 20) permettant d'effectuer ce travail donne une représentation intuitive de la réponse impulsionnelle de la salle et permet d'écouter en temps réel la qualité acoustique décrite, sur un signal sonore donné.

Une fois ce travail « bas niveau » effectué, l'auteur va exploiter cette librairie d'effets pour composer la scène sonore et décrire les propriétés acoustiques de l'environnement qu'il construit. La définition géométrique de cet environnement est donc enrichie d'une description acoustique et le système saura alors à terme, à partir des positions des sources et de la position du point d'écoute, rendre au mieux les propriétés acoustiques souhaitées au travers du son spatialisé ainsi qu'éventuellement les effets d'occlusion ou d'obstruction, lorsque des obstacles séparent la source du point d'écoute.

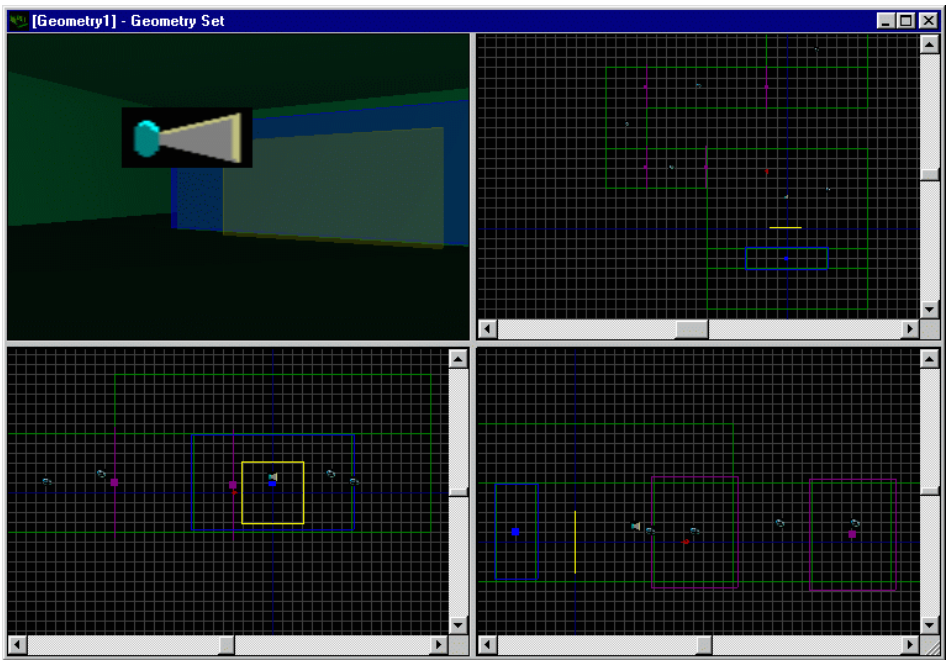


Figure 21 : vue du logiciel E.A.G.LE. (2),
association de propriétés acoustiques à la description géométrique

Nous donnons pour finir, en Figure 21, un aperçu du logiciel EAGLE dans la phase de la description acoustique de l'environnement. Au moyen de représentations multiples, l'auteur peut visualiser la scène, et y incorporer des sources sonores pour les écouter in situ.

2.6 OpenMusic

OpenMusic n'est pas a priori un système de contrôle de la spatialisation : c'est un environnement de programmation visuel dédié à la composition musicale. Concrètement, cet outil sert d'interface au langage LISP, dans sa version orientée objet, et propose à ses utilisateurs un ensemble de classes et de méthodes spécifiquement adaptées à la structuration de données musicales, à leur traitement, et à leur représentation sous forme graphique, au moyen de divers systèmes et conventions de notation. Très ouvert, ce système s'adapte naturellement à différentes sortes d'applications telles que précisément le contrôle de la spatialisation, comme il l'a été montré dans le cadre d'une collaboration entre Carlos Agon et Gilbert Nouno, pour – entre autres – la mise en espace d'une composition de Brian Ferneyhough « Stellae for failed times » pour chœur et électronique [Nouno & Agon, 2002].

Cette étude se concentre sur le contrôle de la librairie spatialisateur IRCAM et l'ensemble de ses paramètres perceptifs et rassemble autour du nom « OmSpat » un ensemble de structures de données et de méthodes générales, adaptées au contrôle de la spatialisation. L'objectif est de générer un ensemble de fichiers « trajectoire » qui seront ensuite utilisés pour traiter en temps différé des sources sonores monaurales à spatialiser à l'aide de la librairie Spatialisateur de l'IRCAM sous l'environnement MAX/MSP.

L'étude porte davantage sur les moyens de décrire simplement des trajectoires compliquées à l'aide de la puissance algorithmique d'OpenMusic que sur l'observation des relations spatiales qu'établissent les sources sonores entre elles. Le système permet cependant d'organiser l'espace dans le temps en produisant pour chacune des sections de la partition des descriptions spatiales appropriées (voir Figure 22).

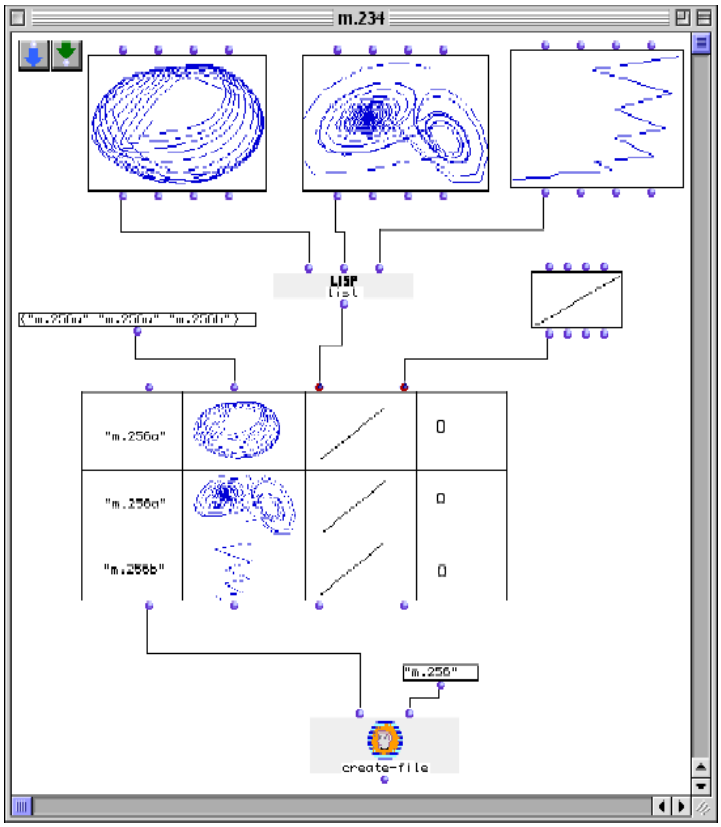


Figure 22 : construction de trajectoires sonores en OpenMusic

OpenMusic sera encore évoqué en annexe de ce document, à l'occasion de la présentation du projet OpenSpace, dont l'objectif était de combiner les projets OpenMusic et MusicSpace afin de mettre en commun les possibilités d'abstraction et de calcul de l'un, avec les avantages du contrôle temps-réel sous contraintes de l'autre.

Chapitre 3
Programmation par contraintes

Une contrainte est une relation, au sens mathématique que l'on souhaite à tout instant vérifiée. La programmation par contrainte consiste à aborder un problème au moyen de contraintes, d'une part en le décrivant complètement sous forme d'un ensemble de ces relations et d'autre part en produisant des algorithmes permettant d'aboutir rapidement à une solution, c'est-à-dire à un élément qui précisément vérifie toutes les propriétés énoncées.

3.1 Applications de la programmation par contraintes

L'histoire de la programmation par contraintes commence vraisemblablement avec « Sketchpad », de I. Sutherland en 1963 ([Sutherland, 1963]), un projet de dessin interactif permettant à l'utilisateur de construire des figures géométriques à partir de primitives et de contraintes. Ce projet, qui innove doublement puisqu'il amène simultanément les idées d'interface graphique et de programmation par contraintes voit son successeur en ThingLab ([Borning, 1981]) qui apporte l'idée supplémentaire de compilation de contraintes en un « plan » de calcul.

La programmation par contraintes a été largement étudiée et mise en œuvre dans nombre de domaines d'applications. Elle a fait ses preuves en particulier dans le cadre de problèmes difficiles à formaliser autrement que précisément par l'intermédiaire d'un ensemble de conditions à vérifier. Ces problèmes sont par exemple de complexité combinatoire et consistent pour les plus connus à placer des reines sur un échiquier, ou résoudre des cryptogrammes.

Nous citons dans les sous-sections suivantes un ensemble d'exemples d'applications de la programmation par contraintes proches du domaine qui nous concerne, avant de nous intéresser aux techniques sous-jacentes.

3.1.1 Les applications multimédia

Plus proches de nos préoccupations, la littérature propose de nombreuses utilisations des contraintes dans les applications multimédia. Un des problèmes posés dans ce contexte est la mise en page de documents multimédia constitués de différents composants graphiques tels que des éléments textuels ou des images. L'utilisation des contraintes dans ce type d'application permet de décrire le document à partir de relations qu'entretiennent ces différents éléments entre eux, et de laisser le système construire le rendu final du document à partir par exemple du format possible de la mise en page. Le lecteur intéressé pourra trouver un panorama très complet des applications de la programmation par contrainte dans les applications multimedia et conception assistée par ordinateur dans [Hower & Winfried, 1996].

Dans « Constraints for the web » ([Borning & Al, 1997]), ces problèmes sont mis en évidence spécifiquement dans le contexte de la construction de pages Web. Le projet présenté s'articule autour d'une architecture dans laquelle les contraintes peuvent être assignées – éventuellement à des degrés de priorité différents – aussi bien par l'auteur du document que par le système de présentation sur la mise en page du résultat. Le rendu final est ainsi un compromis entre les contraintes émises par le système auteur et celle du navigateur. Les variables présentées concernent toutes les dimensions des éléments média mis en jeu. Une figure, par exemple, sera contrôlée par son abscisse gauche et droite, sa largeur, son ordonnée haut et bas, et sa hauteur. Interviennent également les dimensions de la page ainsi que celles des espaces entre colonnes, saut de lignes et marges diverses. Les contraintes proposées sont des systèmes d'équations ou d'inéquations linéaires, portant sur ces variables. Par exemple, une contrainte peut spécifier que la largeur de la page est la somme de la largeur de ses colonnes, des espaces entre colonnes et des marges gauche et droite. Un exemple de contrainte entre deux objets média serait de stipuler que l'ordonnée supérieure de deux figures doit être la même. Pour finir, un exemple d'inéquation consisterait à requérir que la largeur d'une figure soit plus petite que celle de la colonne du document dans laquelle elle apparaît.

Plusieurs architectures sont présentées dans ce travail, autorisant toujours le contrôle total du document coté auteur, mais proposant plus ou moins de flexibilité au niveau du navigateur. Néanmoins ces architectures s'articulent toutes autour de deux algorithmes dits de propagation locale (voir section 3.2) : le premier, « Cassowary » ([Badros & Borning, 1998]), est un algorithme incrémental basé sur la méthode du simplex. Le second « projection-based compilation algorithm » ([Borning & Al, 1997]) traite les mêmes types de contraintes que Cassowary, mais utilise des techniques de compilation pour produire du code java qui sera plus rapidement exécuté sur la machine client. Le premier algorithme permet d'envisager un cadre plus général, et surtout autorise les contraintes du coté navigateur. Le second est plus restrictif puisque les plans de modification des variables sont pré-calculés, mais il est également plus efficace en terme de rapidité. Ces algorithmes seront plus amplement décrits en section 3.4.3 de ce document.

Un autre exemple, le système Madeus ([Sabri-Ismail & Guetari, 1998], [Jourdan & Al., 1998] et [Jourdan & Al., 1998b]) traite également du problème de la construction et présentation de documents multimédia, et s'organise clairement autour d'un moteur de résolution de contraintes. Ce système permet l'édition et la présentation de documents multimédia en utilisant une approche déclarative : le « scénario de présentation » est construit en déclarant d'une part les objets média faisant partie du document et d'autre part les relations temporelles et spatiales qui les relient suivant les quatre dimensions logique, temporelle, spatiale et hypermédia. En plus des objets média simples, Madeus apporte la notion d'objets composites au travers d'une représentation hiérarchique.

Les contraintes sont ainsi posées indifféremment sur les objets simples et objets composites et permettent d'en exprimer la synchronisation spatiale et temporelle, telle que par exemple l'idée que deux extraits vidéo soient centrés verticalement et soit présentés simultanément. Les contraintes temporelles du système sont basées sur les relations d'Allen ([Allen, 1983]) à partir des opérateurs « START », « EQUALS » et « BEFORE ». Les contraintes spatiales, elles, reposent sur les notions standard d'alignement, de centrage et de décalage, autant sur l'axe horizontal que sur l'axe vertical. Le solver de contraintes utilisé, « Deltablue » ([Sannella & Al., 1993b]), sera présenté en section 3.4.1 de ce document.

3.1.2 Le domaine musical

Le domaine musical n'échappe pas aux attraits de la programmation par contraintes. Tout particulièrement, la composition musicale, qui pour certains de ses styles se prête bien à la formalisation, n'a cesse de démontrer ses affinités avec les concepts algorithmiques, que l'approche soit procédurale ou déclarative, c'est-à-dire par exemple au moyen de contraintes.

L'exemple d'application qui a suscité le plus de travaux est vraisemblablement l'exercice d'harmonisation traditionnel. Dans cet exercice, une partie d'un texte musical est donné (il peu s'agir d'un chant, ou d'une voix de basse chiffrée) : l'objectif est de partir de cet « énoncé » ' pour en construire une « réalisation » à n voix (généralement 4 voix pour chœur, ou pour quatuor à cordes), conservant par exemple le chant dans la voix la plus aigue, et remplissant les autres voix en accord avec les règles d'harmonie qui, par exemple pour la plus connue, interdit des mouvements de quintes parallèles entre deux voix. D'autres règles – ou contraintes réellement physiques cette fois ci – portent par exemple sur les limitations physiques en tessiture des chanteurs ou des instruments. Un panorama plus complet des problèmes d'harmonisation musicale sous contraintes peut être trouvé dans [Pachet & Roy, 2001].

Le pionnier dans cette recherche est Kemal Ebcioglu ([Ebcioglu, 1988]) qui, au sein de son projet « Choral » s'intéresse à la réalisation de chorals à quatre voix dans le style précis de Jean-Sébastien Bach. Ce système ne fait pas appel à la programmation par contraintes, mais est fondé sur une approche par règles. Des travaux comparables, cette fois-ci faisant appel à des moteurs de satisfaction de contraintes sont proposés par Philippe Ballesta ([Ballesta, 1998]) qui utilise l'implémentation en langage LISP du moteur de contraintes Ilog-Solver ([Puget, 1994]) et Pierre Roy, qui propose une implémentation de son solver « BackTalk » ([Roy, 1998]) pour l'harmonisation de basses chiffrées ou de mélodies. La Figure 23 donne une illustration de ce travail avec une réalisation à quatre voix de « La Marseillaise ».

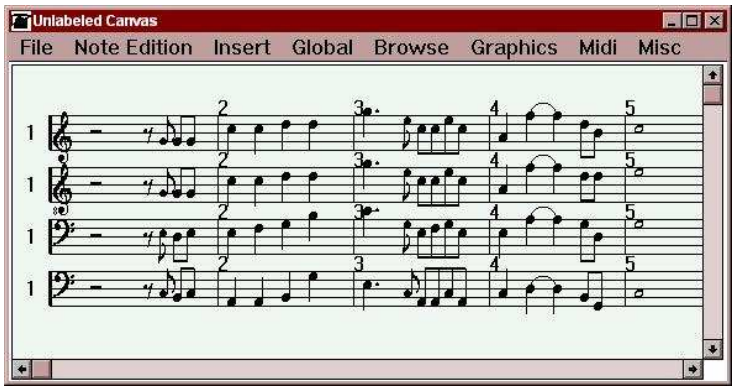


Figure 23: Réalisation à 4 voix des 5 premières mesures de "La Marseillaise" dans le système backtalk

L'équipe « Représentations Musicales » de l'IRCAM produit – entre autres – des environnements ouverts de programmation destinés à l'aide à la composition musicale. Ces logiciels – Patchwork en premier lieu, puis OpenMusic par la suite (voir [Agon & Al, 1998] et [Assayag & Al, 1999]) – apportent aux compositeurs des moyens de représentation et de manipulation de structures musicales adaptées à leurs besoins. Ces deux logiciels ont servi de point de départ à l'implémentation de deux solvers de contraintes particulièrement adaptés au

domaine musical.

Le premier, nommé « PWConstraints » et intégré au logiciel Patchwork est développé par Michael Laurson ([Laurson, 1993]). PWConstraints est un système permettant de résoudre un problème musical en en décrivant les propriétés du résultat sous différents angles. L'utilisateur décrit en premier lieu l'espace de recherche, c'est-à-dire d'une part les variables qui psles valeurs possibles a priori que chacune des variables peuvent prendre. Si le problème consiste à calculer une série de hauteurs par exemple, il est possible par cette manière de définir la tessiture générale du résultat, ainsi que la résolution fréquentielle (par exemple si le problème sera résolu à partir d'une échelle en demi-tons, en quart de tons,…). L'utilisateur défini par ailleurs un ensemble de règles ou contraintes que le résultat doit vérifier. Au système de base, totalement générique s'ajoute un ensemble d'extensions dont par exemple une applications aux problème de recherches dans des séquences polyphoniques : l'utilisateur procure la définition rythmique de la séquence ainsi qu'un ensemble de contraintes portant sur les hauteurs et l'algorithme de recherche tente de remplir ces notes avec des hauteurs satisfaisant les propriétés énoncées par les contraintes.

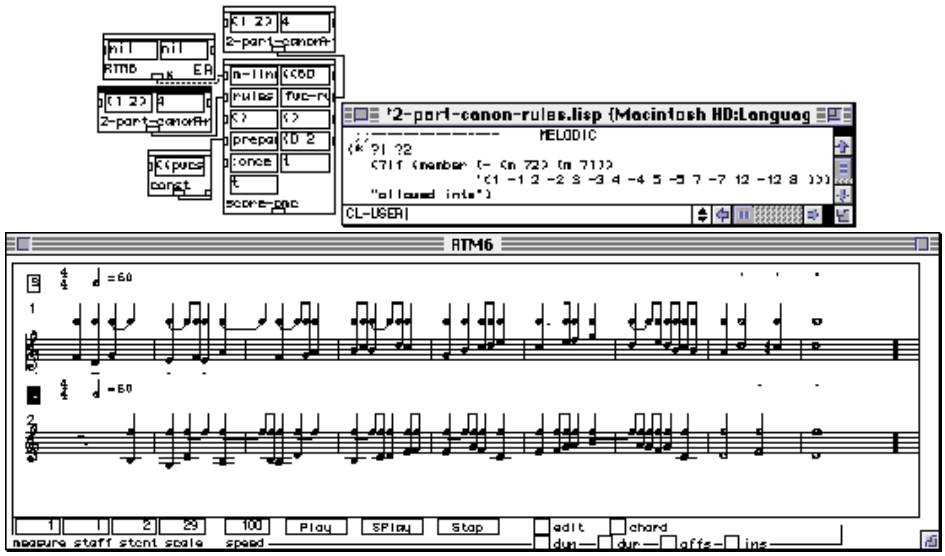


Figure 24 : Construction d'un canon dans PWConstraints

Le second, « Situation », est issu d'une collaboration entre Antoine Bonnet et Camilio Rueda ([Bonnet et Rueda, 1998]). Bien qu'initialement conçu au sein du logiciel PatchWork, il est maintenant intégré sous forme d'une librairie dans OpenMusic. Situation est un langage visuel basé sur les contraintes appliqué à des problèmes harmoniques ou rythmiques permettant de construire des séquences d'accords et des rythmes. Techniquement, le moteur de congtraintes de Situation, « csolver » (voir Figure 25) est basé sur le modèle de la satisfaction de contraintes en utilisant la stratégie dite du « forward checking » (dès qu'une valeur est choisie pour une variable, les domaines des autres variables sont réduits en conséquence).

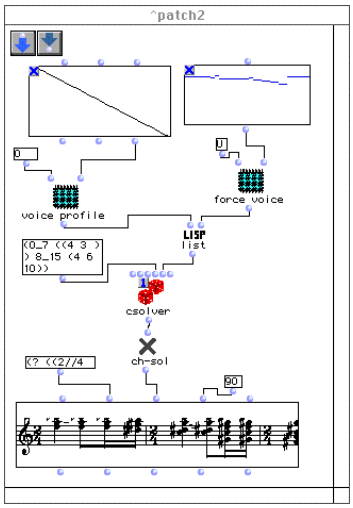


Figure 25 : exemple d'harmonisation automatique de la pièce "Syrinx" de Debussy, à l'aide de Situation dans OpenMusic (extrait de [Agon & Al, 1998])

Ce solver se distingue de systèmes similaires par l'utilisation de mesures de distances permettant d'évaluer à plusieurs niveaux et à chaque instant l'état de la recherche et de guider ainsi le moteur plus rapidement vers une solution.

Les travaux que nous venons d'évoquer sont tous orientés vers la production musicale. Nous mentionnons également ici les travaux de Charlotte Truchet, de l'équipe Représentations Musicale de l'IRCAM dont le but ([Truchet & Al., 2001]) est de proposer au sein de l'environnement OpenMusic, une approche des contraintes suffisamment générale pour pouvoir s'intéresser à toutes sortes de problèmes liés à l'informatique musicale et non seulement à ceux de la production de notes ou de séquences sonores.

Nous retenons plusieurs points intéressants dans cette démarche. Tout d'abord, la description des contraintes d'une part, et du graphe de contraintes d'autre part est réalisée en profitant pleinement des aspects graphiques relatifs à OpenMusic. Une contrainte correspond donc à un « patch » tel que représenté en Figure 26.

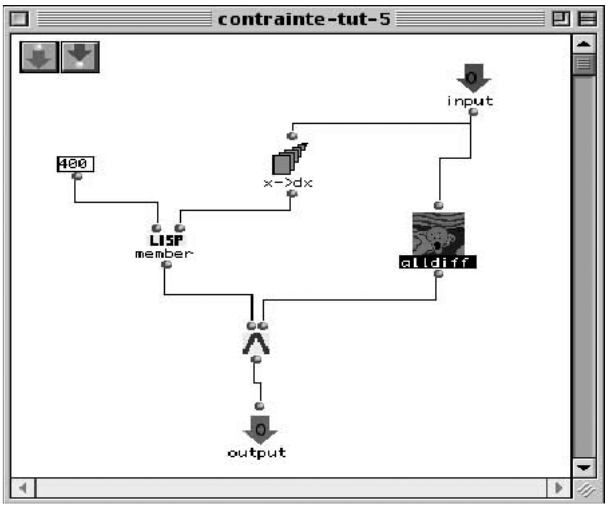


Figure 26 : représentation d'une contrainte sous forme d'un patch

Les utilisateurs peuvent donc conserver le savoir faire en matière de programmation visuelle qu'ils ont déjà acquis pour le mettre à profit dans la conception de contraintes et donc aborder leur démarche compositionnelle sous un angle différent.

Par ailleurs, ce travail a l'originalité d'intégrer deux solvers de contraintes différents : le premier est une approche classique de la satisfaction de contraintes tandis que le second, proposé par Philippe Codognet ([Codognet, 2000]) est un algorithme de recherche locale efficace et bien adapté au domaine musical qui, bien que n'aboutissant pas toujours à la (meilleure) solution, permet de répondre par exemple aux exigences du temps réel : la recherche peut être abandonnée après un délai maximal et la meilleure solution trouvée est délivrée. Ce type d'approche permettrait par exemple à terme de proposer à l'utilisateur une notion de « scalabilité », compromis possible entre les besoins de complétude et / ou de performances. Plusieurs applications illustrent ces travaux. La première consiste à calculer des doigtés à partir de contraintes physiques instrumentales. Il est possible par exemple de chercher des réalisations d'un accord donné à la guitare, comme l'illustre la Figure 27 qui décrit une manière d'obtenir un accord de FA majeur, utilisant la technique du barré.

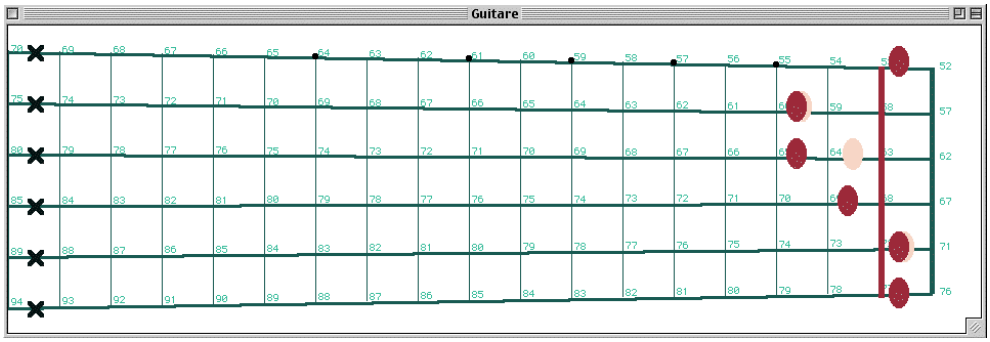


Figure 27 : représentation d'une solution pour le calcul d'un accord à la guitare

Un autre exemple qui s'avère particulièrement original consiste à proposer une application de la programmation par contraintes à l'analyse musicale ([Chemillier & Truchet, 2001],[Chemillier & Al., 2002]). L'objectif de cette approche consiste à modéliser une séquence musicale à analyser par un ensemble de contraintes de sorte que cette séquence soit précisément une des solutions possibles du système. En évitant les solutions de ce problème trop triviales, cette démarche présente l'intérêt de pouvoir générer et examiner des solutions « voisines » de la séquence analysée. Une des séquences sonores analysées vient de la musique traditionnelle de harpe Nzakara (voir Figure 28).

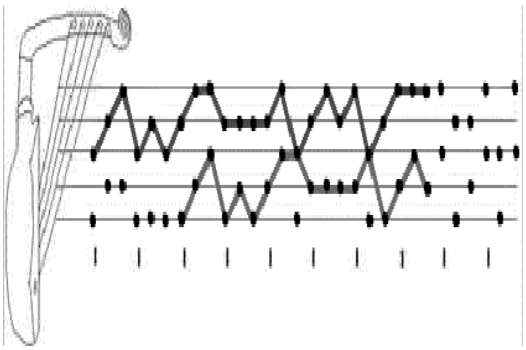


Figure 28 : Analyse musicale de harpe Nzakara

Le problème, mis en évidence initialement par Marc Chemillier ([Chemillier, 1999]) consiste à construire des séquences à deux voix à partir des cinq notes de la harpe ayant des propriétés particulières (la deuxième voix est à une transposition près un canon de la première,…) : cet exemple met particulièrement à l'honneur les algorithmes de recherche adaptative puisqu'il n'existe pas de solution exacte du problème tel qu'il a été formalisé sous contraintes, mais uniquement des solution approchées dont évidement le morceau original de musique traditionnelle.

Pour finir, le domaine temporel a également fait l'objet d'études. Nous citons à titre d'exemple les recherches d'Anthony Beurivé ([Beurivé, 2000]) qui propose dans son logiciel « BOXES » l'utilisation de contraintes pour l'organisation temporelle d'objets sonores. BOXES se rapproche en fonctionnalité des enregistreurs audionumériques, mais propose une approche différente de l'écriture musicale. Une structure hiérarchique générique de « conteneurs » et de « contenus » (objets terminaux de la hiérarchie) est définie. Chacun de ces objets à pour paramètre son temps de départ, son temps de fin et sa durée. Le logiciel s'appuie sur la bibliothèque de contraintes Cassowary ([Badros & Borning, 1998]) qui traite des problèmes à base de systèmes arithmétiques linéaires d'inéquations (cf section 3.4.3 de ce document). Il utilise les trois variables 'début', 'durée' et 'fin' des objets temporels comme variables du système de contraintes et permet donc d'exprimer par exemple des relations d'antériorité ou postériorité temporelle ou encore de déterminer des durées nécessaires pour assurer une synchronisation entre plusieurs flux sonores.

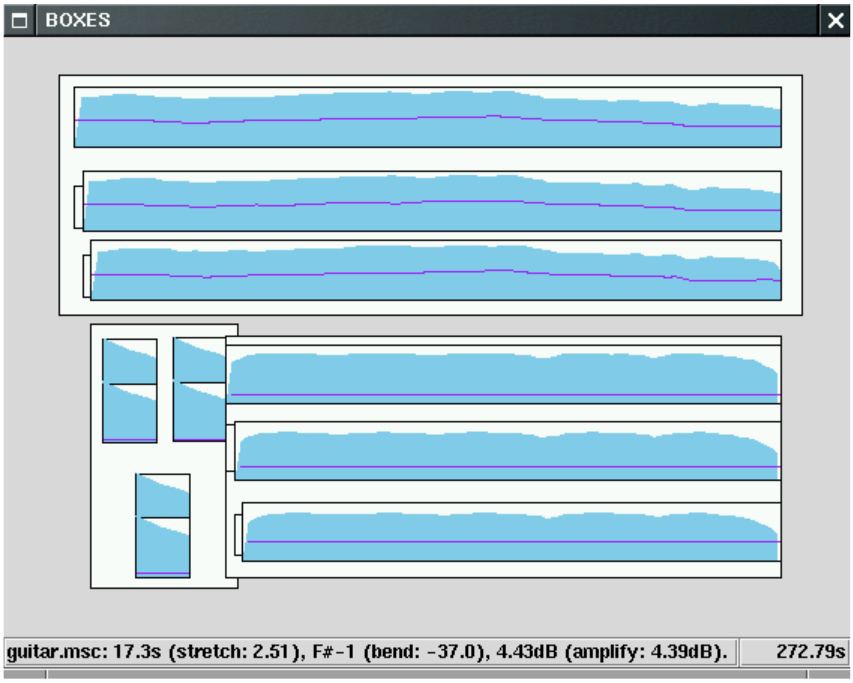


Figure 29 : Aperçu de l'interface graphique du Logiciel BOXES

Des contraintes « intrinsèques » permettent de garantir la cohérence du système. Il s'agit en particulier d'assurer que le début d'une boite se trouve avant sa fin, et que sa durée correspond bien à la différence entre le temps de fin et le temps de début.

Bien que ce projet ne concerne pas le contrôle de la spatialisation, les objets sonores qu'il manipule peuvent être assimilées à des « sources » sonores. BOXES ainsi permet d'établir des relations entre ces sources au moyen des contraintes et se rapproche ainsi fortement de nos objectifs. En outre, les paramètres temporels ne sont pas les seules grandeurs prises en compte dans ce projet : les objets terminaux du graphe, les boites « son », sont caractérisés également par un facteur d'amplitude et une valeur de transposition que l'utilisateur peut également contraindre à sa guise.

Si l'ensemble des applications présentées ici s'approchent de nos préoccupation de part le domaine musical qu'elles abordent spécifiquement, elles en diffèrent tout de même au point de vue technique par le délai accordé au calcul d'une solution. Ces systèmes autorisent en effet un temps de calcul à l'obtention d'une solution qui serait beaucoup trop long pour l'application temp-réel que nous envisageons de contrôle interactif de la spatialisation du son. Dans cette optique la section suivante de ce document s'intéresse à l'utilisation des contraintes au sein des interfaces utilisateurs et – afin d'en assurer la réactivité - prennent en compte le facteur temps dans le calcul des solutions.

3.1.3 Interfaces utilisateur

Nous terminons ce survol d'applications de la programmation par contraintes par le domaine des interfaces utilisateur. Ce domaine nous intéresse particulièrement puisque notre projet en fait partie. Par ailleurs les interfaces utilisateur se devant d'être un minimum réactives vis-à-vis des actions de l'utilisateur, les algorithmes de contraintes employés dans ces projets tiennent compte du facteur temps quand à l'obtention d'un résultat, ce qui n'était pas systématiquement le cas dans les applications présentées précédemment.

L'utilisation des contraintes pour concevoir des interfaces utilisateur est une idée assez développée. Les motivations sont d'après [Myers & Al., 1990] liées au coût de développement et à la difficulté de l'implémentation des interfaces utilisateur : celles-ci doivent en effet répondre à des interactions asynchrones provenant de la souris et du clavier, répondre à ses solicitations en temps réel, gérer simultanément plusieurs fenêtre d'affichage ou encore proposer des représentations graphiques performantes… Les boites à outils à disposition (permettant l'utilisation d'un ensemble de « widgets » standards tels que boutons, ascenseur, zones de saisie,…) semble relativement limitées, complexes, et n'aident que peu le concepteur à concevoir la part la plus importante de son application, ce qui sera représenté dans la fenêtre principale d'une part et la gestion des interactions d'autres part. Enfin, il semble relativement complexe d'ajouter ses propres éléments graphiques à une boite à outil existante…

Fondé sur ces considérations, la boite à outils « Garnet » est donc une première proposition d'utilisation des contraintes pour concevoir des interfaces utilisateur : elle intègre un langage de programmation orientée objet nommé KR, un système de contraintes (originalement baptisé Coral puis abandonné pour être intégré dans KR) gérant les contraintes « one-way » et les cycles, et un système de représentation graphique nommé « Opal » facilitant la tâche de dessin de formes géométriques de base. En particulier Opal gère pour ces objets de base les problèmes de recouvrement et d'actualisation lorsqu'une partie du dessin a été endommagée (par une autre fenêtre par exemple). Garnet propose par ailleurs un ensemble d'« interactors » permettant de gérer les interactions avec l'utilisateur en incorporant un ensemble de « comportements » liés au périphériques d'entrée, comme la souris et le clavier.

Conjointement à Garnet vient un constructeur d'interfaces graphiques nommé « Lapidary » ([VanderZander & Myers, 1991]) qui permet de définir graphiquement l'ensemble des éléments graphiques qui vont constituer l'interface de l'application à concevoir. Lapidary permet également de définir les comportements des objets soit au moyen de menus (pour construire des contraintes par exemple) soit par apprentissage. Pour la définition des contraintes elles-mêmes, Garnet se repose sur le logiciel « C32 », identique à une feuille de calcul et dans lequel il est possible d'entrer des expressions LISP représentant les relations que les contraintes doivent maintenir.

D'autres exemples d'interfaces utilisateur faisant usage des contraintes se trouvent par exemple dans la thèse de Sannella ([Sanella, 1994]) essentiellement orientée autour de l'algorithme « SkyBlue » (voir section 3.4.2 de ce document). Kaleidoscope, par exemple, est un projet intégrant le solver de contraintes SkyBlue et les avantages de la programmation orientée objet impérative : l'utilisateur peut donc utiliser l'un ou l'autre système de programmation en fonction du cas qui se présente.

Nous citons pour finir, CoolDraw ([Freeman-Benson, 1993]), un logiciel basé sur les contraintes pour maintenir des relations géométriques entre des objets graphiques dans un plan bidimensionnel. Il fait appel à une version modifié de l'algorithme SkyBlue pour gérer des relations géométriques telles que des notions d'alignement entre objets ou et des contraintes d'inégalités permettant d'exprimer les notions telles que « à gauche de », « au dessus de », etc.

3.2 Propagation locale et satisfaction de contraintes

Sannella ([Sannella & Al., 1993b]) propose une classification des langages et systèmes fondés sur les contraintes et considère comme primordiale la distinction entre les approches de type « propagation locale », également appelées « modèle par perturbation » et les approrches dites de « satisfaction de contraintes » ou encore « modèle par raffinement ». Si ces différences se manifestent directement dans la façon de concevoir et résoudre les problèmes de contraintes, ces deux approches diffèrent également et principalement dans la donnée initiale du problème qu'elles traitent.

Dans la satisfaction de contraintes, les variables ne sont initialement pas contraintes. Elles disposent toutes d'un « domaine » qui décrit l'ensemble des valeurs qu'elle peuvent prendre : généralement ces domaines sont des ensembles finis de valeurs. Au fur et à mesure que l'algorithme se déploie, les contraintes sont ajoutées une à une et les domaines des différentes variables réduits en conséquence. L'opération qui consiste à réduire les domaines d'une variable s'appelle le filtrage : en principe, l'efficacité d'un algorithme de satisfaction de contraintes repose en grande partie sur l'efficacité du filtrage dont on est capable pour chaque contrainte. Lorsque toutes les contraintes ont été traitées, il suffit de piquer des valeurs arbitrairement dans les domaines réduits pour trouver une solution. Lorsque certains domaines sont vides, le problème n'a pas de solution. Selon le type de problème à résoudre, il est possible de chercher la première solution venue, ou la meilleure solution optimisant un paramètre, ou encore toutes les solutions possibles : avec des domaines finis, le nombre de solutions possibles est normalement lui aussi fini. Cette approche a été adoptée de manière plus ou moins universelle dans la communauté de programmation logique dans le cadre des langages de programmation logique intégrant la programmation par contraintes tels que CAL, CHIP, CLP(R), CLP(sigma*), HCLP(R), Prolog III et les langages de contraintes concurrentes

([Sannella & Al., 1993b]). Le lecteur souhaitant davantage d'informations sur la satisfaction de contraintes pourra par exemple se reporter à la thèse de Pierre Roy ([Roy, 1998]).

Dans la propagation locale, on se donne également un ensemble de variables ainsi qu'un ensemble de contraintes portant sur ces variables, mais l'on requiert en plus une situation initiale pour laquelle les variables ont des valeurs telles que les contraintes sont toutes (ou en tout cas toutes celles qui ont le niveau de priorité « requise ») satisfaites. On se donne alors une perturbation, c'est-à-dire la modification de la valeur d'une variable par rapport à l'état initial. Le problème consiste alors à calculer des modifications pour les valeurs des autres variables de sorte que les contraintes soient à nouveau satisfaites. Ce modèle est le plus couramment utilisé dans les applications a base de contraintes appliquées aux systèmes graphiques ou aux interfaces utilisateur.

Un certain nombre de raisons nous poussent à écarter la satisfaction de contrainte de notre implémentation. En particulier, les domaines des variables que l'on envisage ne sont pas finis : ce problème est surmontable avec la satisfaction de contraintes, mais nécessite un échantillonnage spatial de la scène auditive ce qui ne correspond plus directement au problème tel qu'il était posé initialement. Par ailleurs, des essais ont été conduis Par Anne Liret et Pierre Roy pour tenter d'implémenter les contraintes de MusicSpace au sein du solver BackJava de Pïerre Roy ([Roy, 1998]) et ne se sont malheureusement pas révélés convaincants. Par ailleurs il est difficile de prédire des temps de calcul avec les algorithmes de satisfaction de contraintes traditionnel ce qui est incompatible avec le temps-réel et ne permet pas de concevoir un système réactif.

Pour finir, le problème que nous posons, exprimé au travers de contraintes géométriques ressemble beaucoup à certains problèmes d'interfaces graphiques présentés ([Maloney, 1991]) et pour lesquels les techniques de propagation locale semblaient particulièrement adaptées. Nous retenons donc cette branche de la programmation par contraintes, et présentons dans la section suivante de ce document les distinctions importantes entre les contraintes qui donnent lieu à des types d'algorithmes différents.

3.3 Notions sur les contraintes

Les différents types de relations qu'établissent les contraintes entre les variables, le nombre de variables auxquelles elles s'adressent et les différentes manières d'aboutir à une solution génèrent une taxonomie des contraintes qui se répercute au niveau des algorithmes et moteurs de résolution.

Nous rappelons dans les sections suivantes les notions de base sur les contraintes, essentielles dans le cadre des algorithmes de perturbation, ainsi que les différentes distinctions qu'il est nécessaire de faire entre celles-ci. Nous présentons ensuite un éventail d'algorithmes conçus généralement dans le cadre du modèle par perturbation ou tout au moins adaptés aux applications liées aux interfaces utilisateur, en regard de ces différents types de contraintes.

3.3.1 Notions de base

Un problème de contraintes est la donnée d'un ensemble de variables, de valeurs possibles pour ces variables ainsi que de relations (contraintes) portant sur ces variables. Résoudre ce problème consiste à déterminer des valeurs pour les variables qui satisfassent toutes les

contraintes. Les variables peuvent être à valeurs discrètes (les jours de la semaine, les cases sur un échiquier,…), ou à valeurs continues (un nombre réel compris entre 0 et 1). Conjointement les domaines des variables (l'ensemble des valeurs que ces variables peuvent prendre) peuvent être finis ou infinis.

Les problèmes de contraintes sont généralement représentés sous la forme de graphe biparti tel que représenté en Figure 30. Les sommets de ces graphes sont donc de deux natures différentes, les variables d'une part étiquetées généralement de la forme « V_i » et les contraintes d'autre part marquées « C_j ». Les arêtes de ces graphes permettent alors de relier les contraintes aux variables : il ne peut y avoir d'arête entre deux variables ou entre deux contraintes.

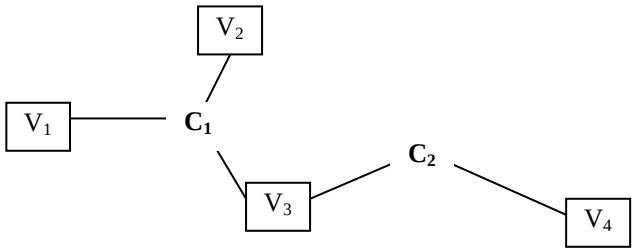


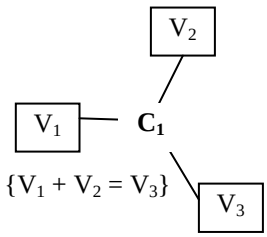
Figure 30 : exemple de graphe de contraintes, composé de quatre variables V_1 à V_4 et deux contraintes C_1 et C_2

La Figure 30 décrit un exemple simple de graphe de contraintes : quatre variables V_1, V_2, V_3 et V_4 sont reliées par deux contraintes C_1 et C_2, qui imposent des relations sur V_1, V_2, V_3 pour la première et V_3, V_4 pour la seconde.

Cette représentation intuitive et relativement synthétique des relations mises en jeu dans un système sera largement exploitée dans notre prototype, MusicSpace

Dans le cadre des algorithmes de perturbation, les variables ont initialement des valeurs telles que chaque contrainte est satisfaite. Une « perturbation » est portée au système – par l'utilisateur par exemple – en changeant la valeur d'une des variables : l'exercice consiste alors à propager cette perturbation au sein du graphe de contraintes de sorte à calculer des nouvelles valeurs pour les variables, afin qu'au final les contraintes soient à nouveau satisfaites.

Les méthodes de résolution associées aux contraintes désignent des composants procéduraux permettant précisément de calculer des nouvelles valeurs aux variables à partir de valeurs perturbées.



Méthodes possibles de résolution :

1. $V_1 \leftarrow V_3 - V_2$
2. $V_2 \leftarrow V_3 - V_1$
3. $V_1 \leftarrow V_3 / 2$; $V_2 \leftarrow V_3/2$
4. $V_2 \leftarrow V_1/4$; $V_3 \leftarrow 5\ V_1 / 4$

…

Figure 31 : méthodes possibles de résolution d'une contrainte

La Figure 31 désigne une contrainte C_1 portant sur les trois variables V_1, V_2 et V_3 et exprimant la relation « $V_1 + V_2 = V_3$ ». Il existe une infinité de manières de résoudre cette équation. Nous en donnons ici quelques exemples. La première méthode consiste à affecter à V_1 la différence des valeurs de V_3 et V_2. La deuxième procède de manière similaire mais propose une valeur pour V_2, à partir des valeurs de V_3 et V_1.

Cette première distinction entre les types de méthodes de résolution des contraintes amène naturellement à considérer les notions de variable d'entrée et de variable de sortie : les variables d'entrée sont les variables dont on utilise les valeurs comme paramètres, et les variables de sorties, celles pour lesquelles une nouvelle valeur est calculée comme résultat de l'application de la méthode. Par exemple, la première méthode de la Figure 31 possède deux variables d'entrée V_3 et V_2 et une variable de sortie, V_1.

Les méthodes de résolution 3 et 4 de la Figure 31 diffèrent des deux premières précisément dans le nombre de variables de sortie. La troisième méthode, par exemple, propose des nouvelles valeurs pour les variables V_1 et V_2 à partir de la valeur de V_3. Cette méthode, possédant plusieurs variables de sortie est dite « multi-output ».

La section 3.3.2 de ce document expose les différences entre les algorithmes de contraintes qui n'admettent qu'une seule méthode par contrainte (contraintes « one-way ») et ceux qui admettent les contraintes ayant plusieurs méthodes de résolution (contraintes « multi-way »).

Nous observerons ensuite, au paragraphe 3.3.3 les différences résultant de l'utilisation de méthodes qui ne possèdent qu'une variable de sortie par rapport aux méthodes en ayant plusieurs.

3.3.2 Contraintes « One-way » et contraintes « Multi-way »

Selon Sannella ([Sannella, 1993]), la différence entre les contraintes unidirectionnelles (one-way ou single-way) et les contriantes multidirectionnelles (multi-way) correspond à la principale distinction entre les algorithmes de perturbation.

Les contraintes unidirectionnelles ne disposent que d'une seule méthode de résolution : la contrainte ne peut donc être satisfaite que d'une seule manière, en calculant la valeur des variables de sortie à partir des valeurs des variables d'entrée. En revanche, dans le cadre des algorithmes de type multi-way, les contraintes ont la possibilité de posséder plusieurs méthodes que le moteur de résolution peut utiliser à sa guise. La contrainte peut donc être satisfaites de différentes manières, en fonction des choix effectués dans la phase de planification.

Au niveau de la représentation graphique du graphe de contrainte, cette distinction entre les contraintes one-way et les contraintes multi-way, et surtout les méthodes correspondantes ainsi que leur sens de calcul peut être ajouté au moyen d'icônes située à coté de chacune des contraintes. Ces icônes reprennent les arêtes des contraintes et y ajoutent des flèches pour déterminer la ou les variables de sorties. Une icône est ajoutée pour chacune des méthodes de résolution que possède la contrainte.

Un exemple est donné en Figure 32 : deux contraintes sont représentées, C_1 et C_2. La contrainte C_2 est dite « one-way » puisqu'elle ne possède qu'une seule méthode de résolution : sa variable d'entrée est V_3, sa variable de sortie V_4. La contrainte C_1, en revanche, est multi-way puisqu'elle dispose de deux méthodes de résolution. La première considère V_2 et V_3 comme

variables d'entrée, et V_1 comme variable de sortie. La deuxième a pour variables d'entrée V_1 et V_2 et pour variable de sortie V_3.

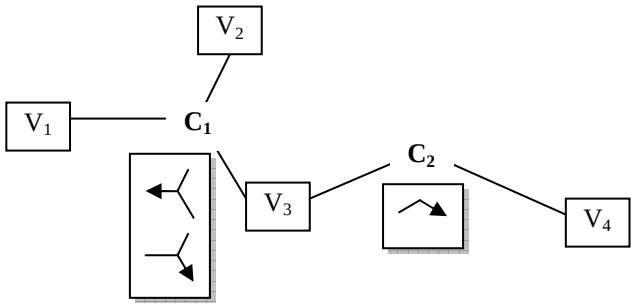


Figure 32 : représentation graphique de contraintes « one-way » (C_2) et de contraintes « multi-way » (C_1)

Selon Sannella ([Sannella & Al., 1993b]), un système one-way peut aisément être simulé par un système multi-way : chaque contrainte one-way est remplacée par une contrainte multi-way munie des annotations « read-only » correspondantes. La simulation inverse est possible également du point de vue théorique : chaque contrainte multi-way s'exprime par un ensemble de contraintes one-way correspondant aux différentes méthodes, mais l'efficacité de son implémentation variera de manière importante suivant le moteur de contraintes utilisé (la transformation correspondante introduit un nombre important de cycles dans le graphe de contraintes).

Un des arguments ([Trombettoni, 1997b]) jouant raisonnablement en faveur des systèmes de contrainte one-way a lieu à la prédictibilité du résultat : à la différence d'un graphe de contraintes one-way, un graphe de contraintes multi-way peut être sous contraint, ce qui signifie qu'il accepte plusieurs solutions. Il est alors généralement difficile de prédire quelle solution va être choisie par le solver. Au contraire, dans les systèmes de contraintes one-way les plus simples, il ne peut exister qu'une seule solution. Il n'y a donc jamais d'ambiguïté.

Malheureusement, les systèmes one-way plus puissants, acceptant par exemple la définition de contraintes partageant la même variable de sortie doivent alors faire face à un problème identique en choisissant la méthode à exécuter, et renvoient aux même défauts de prédictibilité. De plus, on peut préférer [San93b] les systèmes multi-way aux systèmes one-way pour des raisons d'implémentation : il est plus agréable de représenter une relation au sein d'un seul objet plutôt qu'à travers tout un ensemble d'objets. Finalement un dernier argument en faveur des systèmes de contraintes multi-way est que les système one-way ne savent pas prendre ne compte les hiérarchies de contrainte : il faudrait ([Sanella, 1994]) leur ajouter une phase de planification ce qui évidement remet en cause leur efficacité et leur prédictibilité.

3.3.3 Contraintes « single-output » et contraintes « multi-output »

Une autre distinction importante des différents types de contraintes s'effectue suivant le nombre de variables de sorties que peuvent avoir les méthodes de propagation des contraintes n-aires (de manière évidente le problème ne se pose pas dans le cadre des contraintes binaires). Une méthode de propagation est dite « single-output » lorsqu'elle n'a d'effet que sur une seule

variable de sortie de la contrainte correspondante. Au contraire, une méthode de propagation agissant sur plusieurs variables de sortie est dite « multi-output ». Par extension, une contrainte est dite single-output lorsque toutes ses méthodes sont single-output, et multi-output dans le cas contraire.

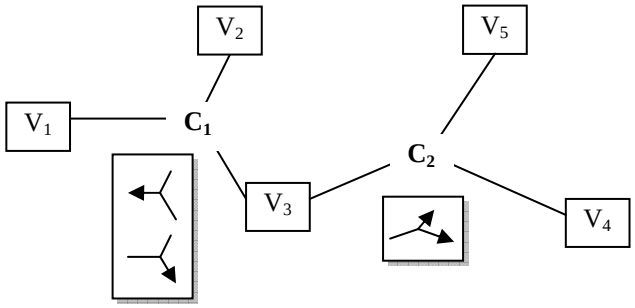


Figure 33 : représentation graphique de contraintes « single-output » (C_2) et de contraintes « multi-output » (C_1)

La Figure 33 donne deux exemples de contraintes : la première, C_1, est dite « single-output » car les méthodes dont elle dispose pour répondre à des perturbations de ses variables d'entrée n'ont d'effet que sur une seule de ses variables de sortie à la fois. En revanche la contrainte C_2 est dite « multi-output » car une perturbation de son unique variable d'entrée V_3 aura un effet simultanément sur les variables de sorties V_4 et V_5.

Les contraintes gérant plusieurs variables de sorties sont particulièrement pratiques dans les applications graphiques. L'exemple classique est celui du maintien d'une même information dans différents systèmes de coordonnées ([Maloney, 1991]) : si x et y représentent les valeurs des coordonnées cartésiennes d'un objet et ϱ et θ les valeurs des coordonnées polaires, il est commode pour maintenir ces deux représentation simultanément d'utiliser des contraintes qui calculent ϱ et θ pour toute modification de x ou y, et qui calcule x et y pour toute modification de ϱ ou θ.

Le problème des correspondances entre les représentations cartésienne et polaire évoqué ci-dessus aurait pourtant pu être abordé au moyen de contraintes à variable de sortie unique en divisant chaque contrainte à sortie multiple en autant de contraintes single-output. Ce formalisme n'a donc a priori pour but principal que de simplifier la modélisation du problème.

3.3.4 Notions sur les cycles et les conflits

Les notions de cycle et de conflit sont liées à la sélection des méthodes de résolution. Cette phase, typique des algorithmes de planification, consiste à sélectionner une méthode de résolution par contrainte parmi les méthodes disponibles. Ces méthodes sont alors confiées à la phase d'exécution pour être ensuite appliquées.

On parle de cycle ou de circuit dans un graphe de méthodes lorsque l'application successive des méthodes de résolution sélectionnées abouti à affecter une nouvelle valeur à une variable qui a auparavant servi de variable d'entrée. La Figure 34 donne un exemple de graphe de méthodes cyclique : si l'ordre d'exécution des méthodes est $C_1 - C_3 - C_2$ par exemple, la variable V1 est initialement utilisée comme variable d'entrée pour le calcul de V_3 dans la

méthode correspondante de C_1, puis, après application de l'unique méthode de C_3, devient variable de sortie pour la méthode sélectionnée pour C_2.

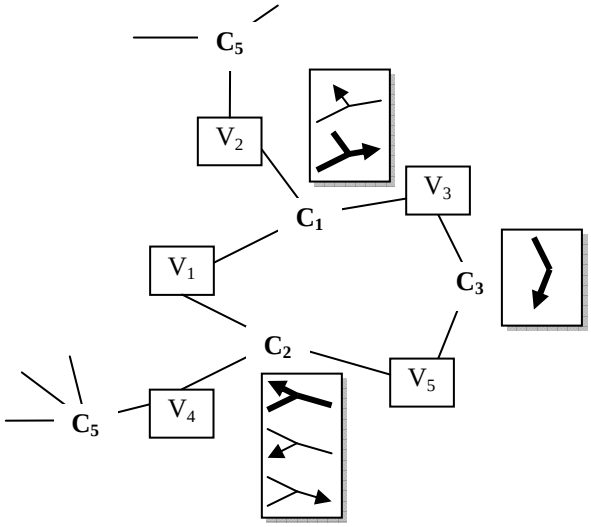


Figure 34 : exemple de cycle dans le graphe de méthodes
(Les méthodes sélectionnées sont représentées en double épaisseur).

Ce type de situation n'est pas très adapté aux algorithmes de propagation : une telle sélection de méthodes lors de la phase de planification provoquerait un bouclage lors de la phase d'exécution des méthodes.

Il existe dans l'exemple de la Figure 34 d'autres sélections de méthodes possibles qui permettent de satisfaire toutes les contraintes sans tomber dans un problème de cycle. De manière générale, le graphe de contraintes est dit « potentiellement cyclique » lorsqu'il existe au moins une sélection de méthodes de résolution qui aboutisse à un problème de cycle.

La notion de conflit, en revanche, désigne le cas dans lequel une variable est désignée variable de sortie pour plusieurs méthodes de résolution sélectionnées. La Figure 35 donne un exemple simple de conflit dans lequel la variable V_5 est variable de sortie des méthodes sélectionnées pour les contraintes C_2 et C_3. Ce type de situation pose également des problèmes de cohérence lors de la phase d'exécution des méthodes.

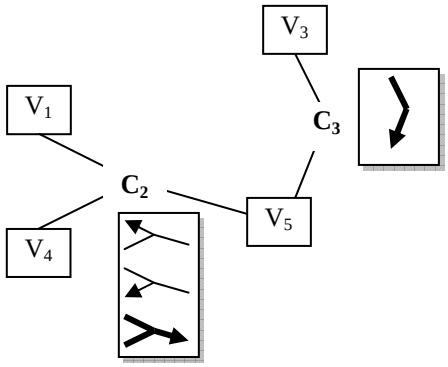


Figure 35 : exemple de conflit dans le graphe de méthodes

Ces deux notions de cycle et de conflit sont importantes dans les algorithmes de planification et seront abordées à nouveau dans les sections 3.4.1 et 3.4.2 de ce document, à l'occasion de la description des solvers de contraintes DeltaBlue et SkyBlue.

3.3.5 Hiérarchies de contraintes :

Selon Sannella ([Sannella, 1993]) les algorithmes de type multi-way sont plus puissants que les algorithmes de type one-way, cependant il peut sembler difficile à l'utilisateur de comprendre et contrôler la manière dont ils produisent les solutions. Par exemple [San93b], si on donne la contrainte A + B = C, et que l'on modifie la valeur de A, faut-il modifier la valeur de B ? ou la valeur de C ? ou bien encore celles de B et de C ? Ou bien interdire de modifier la valeur de A ? … Les premiers systèmes faisaient appel à un ensemble d'heuristiques pour prendre cette décision. Dans ThingLab ([Borning, 1981]), par exemple, les méthodes de propagation sont ordonnées de manière à ce que celles qui doivent être utilisées en priorité apparaissent plus tôt dans cette classification. Dans Magritte ([Gosling, 1983]) l'heuristique consiste à modifier le moins possible de variables en réponse à une perturbation. Ces exemples ont soulevé deux problèmes importants : il est d'une part parfois difficile de prédire comment le système va réagir, et d'autre part, il n'est pas possible de modifier ces heuristiques dynamiquement, celles-ci étant ancrées dans le code.

Les hiérarchies de contraintes ([Borning & Al., 1992]) ont donc été originalement conçues afin de permettre de spécifier de manière déclarative ce qui doit être modifié lorsqu'un système de contraintes est perturbé : l'ensemble des contraintes est organisé sous la forme d'une hiérarchie en affectant une valeur de poids à chacune d'elle, et indiquant ainsi son importance relative par rapport aux autres contraintes. Les hiérarchies de contraintes peuvent donc, par exemple, être utilisées de manière à spécifier de manière déclarative, quelle solution devrait être produite : ceci est indépendant de l'implémentation du moteur de résolution.

Pratiquement, les hiérarchies de contraintes sont construites en définissant un classement entre les contraintes. Un moyen simple est l'ajout d'une propriété supplémentaire à chaque contrainte déterminant son poids et prenant par exemple une des trois valeurs rencontrées typiquement dans la littérature : « weak », « strong », et « required ». D'autres implantations équivalentes sont également possibles et le choix de l'une d'entre elles repose sur le type de manipulations que l'utilisateur est amené éventuellement à effectuer sur ces hiérarchies.

On trouve également d'autres applications aux hiérarchies de contraintes, en particulier ([Sannella & Al., 1993b]) dans les algorithmes effectuant une phase de planning. Dans le cas de problèmes sur-contraints , lorsque toutes les contraintes ne peuvent pas être satisfaites simultanément, celles dont la priorité (le niveau hiérarchique) est la plus basse sont enlevées progressivement jusqu'à ce qu'une solution puisse être trouvée satisfaisant aux maximum les contraintes de priorité élevée.

Le problème de prédictibilité que l'on peut rencontrer dans les systèmes de contraintes est partiellement résolu au moyen de contraintes dites « de maintien » (« stay constraints »). Ces contraintes unaires ne servent qu'à spécifier qu'une variable doit conserver sa valeur actuelle et sont munies de la priorité la plus faible de sorte qu'elles ne soient pas un obstacle aux autres contraintes de priorité plus fortes. Chaque variable du graphe est ainsi munie (parfois implicitement) d'une telle contrainte de sorte que la solution à laquelle le système abouti est telle que les variables qui n'avaient pas explicitement besoin de changer de valeur ont conservé

leur valeur initiale. L'exemple typiquement rencontré ([Helm & Al., 1995]) pour illustrer cette notion concerne l'utilisation des contraintes pour effectuer la mise en page d'un document contenant un ensemble d'objets graphiques (texte, image,…) repérés en abscisse, ordonnée, largeur et hauteur : une contrainte d'alignement vertical par exemple entre un ensemble d'objets graphiques produit une valeur en abscisse similaire pour tous les objets contraints. Les contraintes de maintien servent alors à ce que les valeurs en ordonnée, largeur et hauteur ne varient pas.

3.3.6 Annotations « Read Only »

Dans le cadre le plus général, une contrainte représente une relation et peut donc a priori permettre de calculer une valeur pour n'importe laquelle de ses variables à partir des valeurs de ses autres variables. Il existe un formalisme permettant pour certains solvers de spécifier qu'une variable ne doit pas être utilisée comme variable de sortie d'une contrainte : cette variable est alors considérée : « read only ».

Sanella donne un exemple de cette particularité dans [Sannella & Al., 1993b] : dans la contrainte A? +B? = C?, les variables A et B ont toutes deux été marquée comme « read-only » de telle sorte que le moteur de contrainte ne peut modifier que la variable C pour satisfaire cette contrainte.

Ce formalisme n'est néanmoins qu'un mode économique de spécifier quelles sont les différentes méthodes de résolution d'une contrainte lorsque qu'il est plus efficace de le faire simplement en spécifiant que certaines variables sont en lecture seule.

3.3.7 Contraintes fonctionnelles

La distinction entre les contraintes qui sont fonctionnelles et celles qui ne le sont pas est importante en regard des techniques sous-jacentes de résolution. Une contrainte est dite fonctionnelle ([Borning, & Al., 1996]) lorsque pour chacune de ses variables contraintes, qui n'est pas marquée comme « read-only », il existe une valeur unique pour cette variable qui satisfasse la contrainte, étant donné les valeurs des autres contraintes.

Typiquement la relation : A + B = C est fonctionnelle : chaque variable peut être complètement déterminée de façon unique étant données les valeurs des autres variables. Au contraire, un exemple de contrainte non fonctionnelle est par exemple une inégalité : étant donnée la relation A < B et une valeur quelconque pour B, il n'est pas possible de définir A de manière unique.

La prise en compte ou non des contraintes d'inégalités – ainsi que de la manière dont elles sont gérées – est un paramètre important des algorithmes de perturbation. Dans les cas les plus complexe, un solver spécifique est utilisé pour s'occuper des parties du graphe dans lesquelles se trouvent les contraintes non fonctionnelles et communique avec les autres solver par l'intermédiaire d'un algorithme de plus large envergure (voir section 3.4.8 de ce document).

3.4 Algorithmes fondamentaux :

Nous présentons dans cette section un ensemble de recherches propres aux algorithmes de propagation locale. Ces travaux sont en majeure partie issus des recherches menées au

département d'informatique et sciences de l'ingénieur de l'université de Washington dans l'équipe d'Allan Borning où un ensemble de solvers fondés sur la propagation locale ont été conçus pour répondre à différents problèmes de contraintes.

3.4.1 DeltaBlue :

DeltaBlue est le premier d'une série d'algorithmes développés au sein de l'université de Washington. Il est originalement conçu par Freeman-Benson ([Freeman-Benson & Maloney, 1989],[Freeman-Benson & Al., 1990]) et largement détaillé dans les publications suivantes : [Maloney, 1991] et [Sannella & Al., 1993b]. DeltaBlue un algorithme incrémental : il permet donc d'ajouter ou retirer des contraintes facilement en cours de d'exécution, et lorsqu'une telle opération a lieu, de ne pas avoir à recommencer l'ensemble des calculs depuis l'état initial. Cet algorithme permet de résoudre les problèmes de hiérarchie de contraintes multi-way à l'aide de la propagation locale. Il est typiquement étudié pour permettre la construction d'interfaces utilisateur interactives. Deux restrictions importantes s'appliquent cependant à cet algorithme : tout d'abord DeltaBlue ne sait pas gérer les cycles dans le graphe de contraintes. Lorsque DeltaBlue rencontre un cycle, une erreur est signalée et l'algorithme s'arrête. Par ailleurs, DeltaBlue ne sait gérer que les méthodes n'ayant qu'une seule variable de sortie. Cette restriction limite ainsi les possibilités d'interaction entre DeltaBlue et d'autres moteurs de contraintes plus performants.

DeltaBlue procède en deux phases : la phase de planning et la phase d'exécution. La phase de planning consiste à sélectionner pour chaque contrainte, la méthode qui doit – si nécessaire – être utilisée pour répondre à une perturbation donnée. Une fois cette première étape réalisée, la phase d'exécution se contente simplement d'appliquer les méthodes sélectionnées, et de propager les valeurs obtenues. Pour un graphe de contraintes donné, DeltaBlue construit donc un « plan » permettant de re-satisfaire les contraintes suite à une perturbation. Ce plan peut alors être utilisé a volonté avec différentes valeurs de perturbation, jusquà ce qu'intervienne une modification du graphe de contraintes : cet aspect est particulièrement intéressant au niveau des performances, en particulier dans le cadre d'applications graphiques. En effet, dans ce type d'application, une perturbation provient typiquement d'une action de l'utilisateur, par exemple au moyen de la souris : il est fréquent que le même paramètre soit modifié un nombre important de fois jusqu'à ce qu'il prenne la valeur souhaitée par l'utilisateur.

Les contraintes dans DeltaBlue sont représentées typiquement par les méthodes associées, qui calculent chacune une valeur pour une variable de sortie, à partir des valeurs de toutes les autres variables. Mais il est trois autres types de contraintes un peu particulières : les « stay constraints » ou contraintes de maintien servent à ce qu'une variable ne soit pas modifiée inutilement de sorte que le résultat de l'algorithme soit relativement prédictible par l'utilisateur. Implicitement, toutes les variables possèdent une contrainte de maintien. Ces contraintes n'ont pas de variable d'entrée et qu'une seule variable de sortie. Les « input constraints » et « edit constraints » servent à apporter les perturbations au systèmes. Les premières sont typiquement reliées aux événements de la souris, tandis que les secondes permettent manuellement d'affecter une valeur à une variable. Pour ce faire, par exemple, une contrainte « edit » est ajoutée à la hiérarchie. Celle-ci impose une valeur à sa variable contrainte, puis est supprimée.

Initialement, la hiérarchie de contrainte est vide. L'ajout ou la suppression de contraintes se fait par l'intermédiaire de deux méthodes : AddConstraint et RemoveConstraint. Lorsqu'une

contrainte est ajoutée ou supprimée, l'algorithme utilise les méthodes de cette contrainte pour affecter aux variables contraintes une valeur « locally-predicate-better ». DeltaBlue gérant les hiérarchies de contraintes, l'objectif de cette phase est de satisfaire en premier lieu le plus grand nombre des contraintes « requises » (de priorité la plus haute) et si possible les contraintes « souhaitées » (de priorité moindre) jusqu'aux contraintes de maintien (de priorité les plus faibles). La thèse de Maloney ([Maloney, 1991]) démontre que les solutions produites par DeltaBlue sont effectivement « locally-predicate-better ».

DeltaBlue mémorise la solution courante sous la forme d'un graphe de solution (« solution graph »), qui décrit la manière de recalculer les valeurs pour les différentes variables. L'idée majeure est d'associer à chaque variable contrainte une information supplémentaire de telle sorte que le graphe de solution puisse être modifié de manière incrémentale en n'analysant qu'une faible proportion de la hiérarchie de contraintes.

3.4.2 SkyBlue :

SkyBlue ([Sannella, 1993]) est un moteur de contraintes « multi-way » qui utilise la propagation locale et gère les hiérarchies de contraintes. Successeur de DeltaBlue, il a pour but de remédier aux deux limitations importantes de DeltaBlue : l'interdiction des cycles dans le graphe de contraintes et la limitation des contraintes à une seule variable de sortie.

Pour pouvoir prendre en compte les contraintes à sorties multiples (multi-output constraints), SkyBlue effectue une recherche avec backtracking afin de construire un arbre des méthodes (method vines) lors de la sélection des méthodes à exécuter. Les contraintes dans SkyBlue sont représentées par leur méthodes : celles-ci sont des procédure qui, à partir des valeurs d'un sous-ensemble des variables (les variables d'entrée) calculent des valeurs pour le reste des variables (les variables de sortie). De la même manière que pour DeltaBlue, l'algorithme consiste dans un premier temps à sélectionner une méthode pour chaque contrainte de sorte à construire un graphe de méthodes (mgraph). Un mgraph est dit sans conflit lorsque chaque variable n'est variable de sortie que d'au plus une seule méthode. Lorsqu'un mgraph est sans conflit ni cycle orienté, il peut être utilisé directement pour résoudre une perturbation.

En présence de cycle, SkyBlue identifie le groupe de méthodes responsables du cycle et le réduit à une méthode unique, dont il confie le calcul à des algorithmes spécialisés (« cycle solvers ») (voir exemple Figure 36). Par exemple, si toutes les contraintes d'un cycle sont des équations linéaires, SkyBlue fera appel à un moteur de cycles utilisant l'élimination Gaussienne pour satisfaire les contraintes de ce cycle, et la propagation locale pour le reste du graphe.

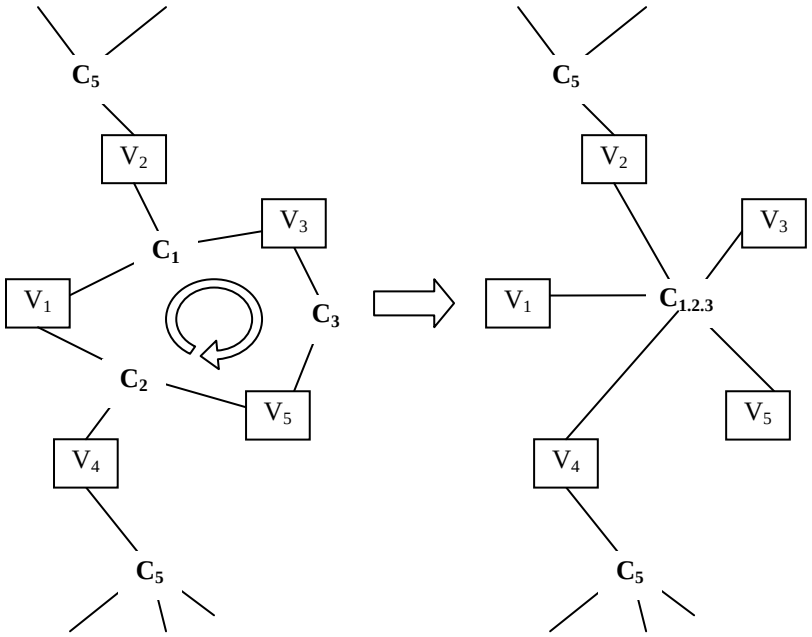


Figure 36 : identification des contraintes (C_1, C_2 et C_3) responsables d'un cycle (gauche) et réduction à une contrainte unique $C_{1.2.3}$ gérée par un moteur de résolution de cycles (droite)

Pour gérer les hiérarchies de contraintes, SkyBlue construits des graphes de méthodes (mgraphs) qui sont « method-graph-better ». Un graphe de méthodes est dit method-graph-better lorsque aucune des méthodes non utilisées pourraient être sélectionnée sans que cela n'introduise de conflit au niveau des contraintes de priorité égale ou supérieure.

Au niveau complexité, l'algorithme de base de SkyBlue ralentit considérablement lorsque le graphe de méthodes devient important. Le problème n'est pas simple, et en particulier [Maloney, 1991] contient la preuve que tenir compte des cycles ainsi que des méthodes multi-output rend le problème NP-Complet. Pour remédier à ces problèmes de temps de calcul, SkyBlue est enrichi d'un ensemble de techniques destiné à améliorer les performances dans le cas de mgraphs important, consistant par exemple à trier les méthodes non sélectionnées.

3.4.3 Cassowary

Cassowary ([Borning & Al, 1997], [Badros & Borning, 1998], [Badros & Borning, 2001]) est un algorithme différent des algorithmes précédent. Il s'intéresse aux contraintes d'égalité et inégalité linéaires en les traitant à partir d'une version incrémentale de l'algorithme de simplex.

Ce type de contraintes semble en effet adapté à un nombre important de problèmes tels que la mise en page de documents ([Borning & Al, 1997]) : dans ce type d'applications, les contraintes servent à exprimer des relations sur les dimensions d'un document ou entre ses différents composants. On note par exemple :

- dimensions des marges et des espaces entre les colonnes principales
- somme des marges espaces et colonne vaut la largeur de la page,…
- une figure doit être à gauche d'une zone de texte, …

Cet algorithme ne craint par définition pas les cycles. De plus les notions de méthodes

énoncées dans les algorithmes précédents n'ont plus de signification : seule la relation exprimée sous forme d'une égalité ou inégalité est utilisée. De même chaque variable peut être indifféremment utilisée comme variable d'entrée ou de sortie.

Cassowary est un algorithme assez répandu et doit sa popularité au fait qu'il réuni un ensemble d'atouts importants : hormis le fait qu'il est accessible, gratuitement, sous la forme d'une librairie partagée précompilée, il semble que cet algorithme propose un compromis intéressant entre simplicité, performance et degré d'expressivité des contraintes utilisables dans le cadre d'applications graphiques interactives. Anthony Beurivé l'a expérimenté dans son projet « BOXES » de gestion temporelle d'événements sonores (voir [Beurivé, 2000]).

3.4.4 Projection Based Compilation

« Project Based Compilation » est un autre algorithme de contraintes présenté en parallèle avec Cassowary dans [Borning & Al, 1997]. Il s'attache à la même classe de problèmes que celui-ci, les systèmes d'équations et inéquations linéaires, toujours dans le cadre de la production de documents multimedia. A la différence de Cassowary, Projection Based Compilation fait usage de techniques de compilation pour produire du code Java qui peut être exécuté sur la machine sur laquelle sera visionnée le document multimédia. Il n'est donc pas nécessaire d'utiliser de moteur de contrainte au moment du rendu : le code exécutable transmis sait maintenir les relations énoncées par les contraintes pour un nombre donné de perturbations. En d'autres termes, le code contient une procédure pour chacune des actions que l'utilisateur peut effectuer à l'aide de la souris. Les détails concernant la technique de compilation utilisée sont donnés dans [Harvey & Al., 1997].

Les avantages de cette approche sont nombreux. En particulier, il n'est pas nécessaire de posséder de solver de contraintes pour visionner le document et agir de manière interactive. Les performances profitent également d'une telle approche. En revanche, l'inconvénient majeur est que le système perd évidement toute possibilité incrémentale : il n'est une fois le graphe de contraintes compilé plus possible ni d'ajouter ni de supprimer des contraintes.

3.4.5 QOCA

QOCA ([Helm & Al., 1995] et [Marriott & Al., 1998]) est une boite à outil construite autour de la programmation par contraintes et spécialement conçue pour les applications graphiques interactives. Ce projet contient trois vrais solvers de contraintes et deux solvers auxiliaires, qui permettent de gérer les contraintes arithmétiques linéaires, à partir de deux différentes mesures de distance, la distance euclidienne au carré et la distance de Manhatten. Les deux premiers solvers sont le « LinEqSolver » et le « LinIneqSolver » qui gèrent respectivement les équations et inéquations linéaires et comparent les solutions au moyen de la distance euclidienne au carré : ils sont basés sur une méthode de pivot. Le troisième solver, nommé « CassSolver » est basé sur l'algorithme Cassowary déjà présenté dans ce document (voir section 3.4.3).

Une originalité importante de cette étude repose sur la notion de « metric space model », qui sur le plan théorique est comparable à celle de hiérarchie de contraintes. Ce modèle maintient en permanence un ensemble de contraintes et une solution courante et permet, au moyen d'une métrique, d'évaluer la distance entre deux ensembles de valeurs pour les variables. Lorsque le système est perturbé, une nouvelle solution est calculée en « suggérant » des nouvelles valeurs pour les variables : parmi les différentes solutions possibles, celle qui est

conservée est celle qui minimise la distance à la solution précédente. QOCA (pour « Quadratic Optimization Constraint Algorithm ») utilise précisément la méthode des moindres carrés pour comparer les solutions entre elles et revient à un problème d'optimisation quadratique dans lequel une fonction (d'erreur) doit être minimisée en respectant un ensemble de contraintes linéaires arithmétiques d'égalité et inégalité.

Un cadre général unifié permet d'intégrer facilement un solver supplémentaire au projet, qui serait capable par exemple de prendre en compte des contraintes de nature différente. Concrètement un solver doit implanter trois méthodes de base dans le mode « manipulation », c'est-à-dire le mode dans lequel l'utilisateur définit l'ensemble des contraintes : ces méthodes sont AddConstraint, RemoveConstraint, et ChangeConstraint et rendent toutes des valeurs booléennes suivant que l'opération correspondante a réussi ou échoué. Le mode « edition » sert à modifier les valeurs des variables d'entrée. C'est donc typiquement un mode d'utilisation pendant lequel le système reçoit des perturbations et fait appel au solver pour rétablir les relations souhaitées. Dans ce mode le solveur présente la méthode SetEditVar(variable) appellée autant de fois que nécessaire pour préciser quelles sont les variables d'entrée, et les méthodes BeginEdit, Resolve et EndEdit. La gestion des variables au niveau des solveur se fait également de façon standardisée en appellant les méthodes AddVar et RemoveVar qui respectivement ajoute ou retire une variable du graphe.

Les solvers intégrés à QOCA sont les suivants :

- LinEqSystem : c'est le plus simple des solvers de QOCA. Il permet de gérer des systèmes d'équations linéaires, est incrémental, mais ne sait pas tenir compte du « metric system model ». Il résout simplement le système d'équations à partir d'une méthode par tableau.

- LinEqSolver : fondé sur LinEqSystem, il gère les contraintes d'égalité linéaires et se sert d'une distance euclidienne (pondérée) au carré pour comparer les solutions entre elles.

- LinIneqSystem : de même que pour LinEqSystem, LinIneqSystem résout les systèmes d'inéquations linéaires, mais ne tient pas compte du « metric system model ». Il sert de sous-solver au LinIneqSolver présenté ci-dessous.

- LinIneqSolver : fondé sur une méthode de pivot (linear complementary pivoting), ce solver utilise également une distance euclidienne au carré pondérée pour comparer les solutions, mais gère les contraintes d'inégalités linéaires.

- Cassowary solver : pour gérer les contraintes d'égalité et inégalité linéaires au moyen d'une version incrémentale de l'algorithme de simplex. Il compare les différentes solutions au moyen d'une distance de Manhatten pondérée (voir section 3.4.3 de ce document).

3.4.6 OTI Constraint Solver

L'ambition liée au solver OTI (Object Technology International, qui a subventionné cette recherche) ([Borning & Freeman-Benson, 1995]) est de produire un moteur de contraintes robuste, facilement utilisable dans le cadre de projets d'applications graphiques interactive, et d'une qualité suffisante – notamment au niveau des performances – pour en permettre

l'exploitation commerciale.

Le cahier des charges de ce projet requiert donc d'une part des performances suffisamment bonnes pour permettre la construction d'interfaces graphiques réactive, et d'autre part une expressivité au niveau des contraintes permettant entres autres de gérer le cas multi-way autant que one-way, les hiérarchies de contraintes (avec les trois niveaux de priorité « soft », « prefered » et « required ») ainsi que les cycles, à condition d'être complété par un moteur de traitement des cycles spécifique. Il en va de même pour les contraintes d'inégalités. Pour finir, au niveau de l'expressivité, les contraintes souhaitées ne se limitent pas à contrôler un nombre limité de types de données (comme des nombres par exemple) mais doivent pouvoir exprimer des relations sur tous types de propriétés d'objets, tels que des couleurs, par exemple, ou des polices de caractères.

L'algorithme retenu et utilisé dans ce projet est « Ultraviolet ». Il est davantage détaillé dans la section 3.4.10 de ce document. Le solver OTI en propose une implémentation en langage SmallTalk dénommée librairie de contrainte ENVY. Ce projet bénéficie également des techniques de compilation de plan (telles que décrite dans la section 3.4.4 de ce document) afin de permettre la distribution de produits basés sur les contraintes, mais n'en incluant pas de solver.

3.4.7 QuickPlan et la propagation de degrés de liberté

QuickPlan ([Vander Zanden, 1996]) est un solver de contraintes incrémental, qui gère les hiérarchies de contraintes mutli-way et « dataflow ». Une contrainte dataflow est définie comme une équation à laquelle sont associées une ou plusieurs méthodes qui permettent de la résoudre. Dans ce paradigme, une méthode consiste en une procédure permettant de calculer des valeurs pour un ensemble (non vide) de variables de sorties, a partir des valeurs d'un ensemble (éventuellement vide) de variables d'entrée.

La raison fondamentale à QuickPlan en regard des autres algorithmes existant s'appuie sur la constatation que dans le cadre des algorithmes incrémentaux traitant les hiérarchies de contraintes rencontrent deux limitations importantes : tout d'abord, ils ne permettent pas de garantir que la solution sera acyclique. Par ailleurs, leur complexité est dans le pire des cas évolue de manière exponentielle avec le nombre de contraintes ([Sanella, 1994] contient la démonstration assurant que SkyBlue est dans le pire des cas de complexité $O(M^N)$ où M est le nombre maximum de méthodes par contraintes et N est le nombre de contraintes). QuickPlan répond à ces limitations en proposant une complexité dans le pire des cas en $O(N^2)$ – où N est le nombre de contraintes – pour toute hiérarchie de contraintes multi-way de type dataflow qui admet au moins une solution acyclique sans conflit.

La technique sous-jacente à QuickPlan est la propagation de degrés de liberté. La seule restriction importante qui s'applique au niveau des contraintes est que les méthodes doivent utiliser toutes les variables d'une contrainte, soit en tant que variable d'entrée, soit en tant que variable de sortie. De la même manière que pour les algorithmes présentés précédemment (DeltaBlue par exemple), QuickPlan opère en deux phases : la première phase prépare un plan d'exécution et la deuxième exécute les méthodes sélectionnées dans le plan. Mais l'originalité de cet algorithme réside dans la phase de planification, dans laquelle est proposée une amélioration de l'algorithme de propagation de degrés de liberté (« Propagation of degrees of freedom » dans [Sutherland, 1963] et [Borning, 1981]) qui en fait une version incrémentale,

sachant gérer les contraintes multi-output, et les hiérarchies de contraintes.

L'algorithme de base de la propagation de degrés de liberté s'applique aux contraintes n'ayant qu'une seule variable de sortie (single-output). Il procède de la manière suivante : on recherche dans le graphe de contraintes une variable « libre » (free variable), c'est-à-dire une variable qui n'est reliée qu'à une seule contrainte, et pour laquelle il existe une méthode telle que cette variable en soit variable de sortie. Cette méthode est alors sélectionnée (pour être plus tard exécutée dans la 2ème phase de l'algorithme) et la variable ainsi que la contrainte sont ôtées du graphe. L'algorithme reprend alors de manière identique sur le sous graphe restant, jusqu'à épuisement des contraintes.

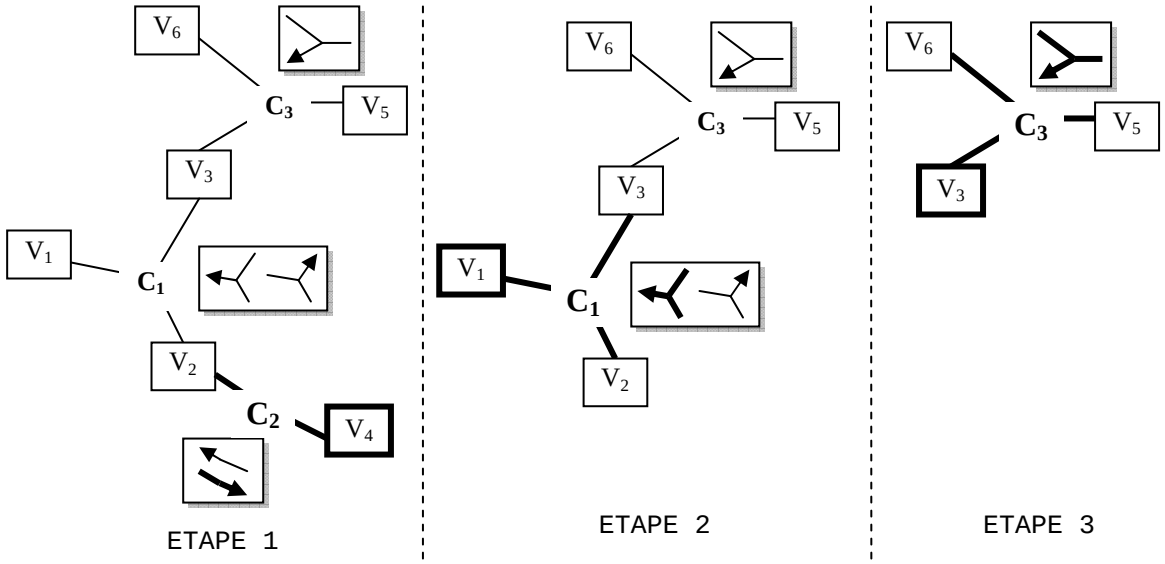


Figure 37 : exemple de phase de planning dans l'algorithme de propagation de degrés de liberté

La Figure 37 donne un exemple simple d'exécution de la phase de planification dans le cadre de la propagation de degrés de liberté. A l'étape 1, une première variable libre est choisie. Dans notre exemple il s'agit de la variableV_4, et de la méthode associée à C_2 pour laquelle V_4 est variable de sortie. Ce choix est arbitraire, et V_1 aurait également pu être choisie puisqu'elle n'est contrainte que par C_1 et que C_1 possède une méthode pour laquelle V_1 est variable de sortie.

V_4 et C_2 sont donc ôtées du graphe et l'algorithme reprend à l'étape 2 sur le sous graphe restant. Cette fois ci il n'y a qu'une seule variable libre disponible, c'est V_1. La méthode de C1 appropriée est donc sélectionnée, puis V1 est supprimée du graphe de contraintes ainsi que C1. On remarque que V2 disparaît également puisque cette variable n'est plus reliée à aucune contrainte.

Finalement à l'étape 3, il ne reste plus qu'une variable libre et méthode à utiliser. Celle-ci est sélectionnée et la phase de planification est terminée. Le résultat est le graphe de méthodes représenté en Figure 38. Les méthodes sélectionnées n'ont alors plus qu'à être exécutées (dans l'ordre inverse de celui dans lequel elles ont été sélectionnées) pour calculer les nouvelles valeurs des variables.

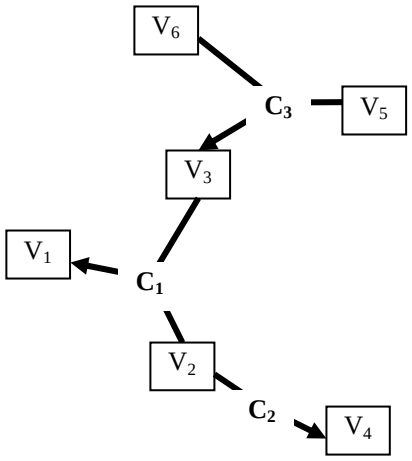


Figure 38 : graphe des méthodes résultant

Afin d'autoriser la prise en compte de contraintes associées à des méthodes ayant plusieurs variables de sortie, l'algorithme ne recherche non plus une, mais un ensemble de variables libres, qui correspondent aux variables de sortie d'une des méthodes de la contraintes correspondante. S'il existe plusieurs méthodes pour cette contrainte, l'algorithme en choisi de préférence une qui possède le plus petit nombre de variables de sortie (cette heuristique permet de minimiser le nombre de variables libres utilisées et donc de maximiser le nombre de contraintes qui seront satisfaites). L'algorithme procède ensuite de manière identique à sa version originale : on sélectionne la méthode en question puis retire les variables de sortie ainsi que la contrainte du graphe et on recommence sur le sous graphe restant.

A terme, l'algorithme s'arrête soit lorsque l'ensemble des contraintes ont été éliminées, soit lorsqu'il n'existe plus assez de variables libres pour choisir une méthode. Dans le premier cas, le graphe des méthodes résultant est acyclique et peut être utilisé tel quel dans la phase d'exécution. Dans le deuxième cas, la portion du graphe restant possède un cycle et doit être confiée à un algorithme de traitement des cycles spécifique.

L'ordre dans lequel sont choisies les variables libres n'a pas d'effet sur l'exactitude du résultat. En revanche, cet ordre peut avoir une incidence sur le résultat lui-même : la notion de hiérarchie de contrainte peut donc aider à guider l'algorithme vers la solution attendue. L'amélioriation proposée pour la prise en compte des hiérarchies de contraintes permet, lorsque toutes les contraintes ne peuvent pas être satisfaites, de désélectionner les contraintes de priorité les plus faibles pour éventuellement permettre à d'autres contraintes de priorité plus forte d'être prises en compte. Lorsque l'algorithme arrive au point où il ne reste plus dans un sous graphe suffisamment de variables libres pour sélectionner une méthode, au lieu de terminer, celui-ci tente de supprimer la contrainte de priorité la plus faible du graphe initial puis essaye de résoudre le nouveau sous graphe. L'algorithme alterne ainsi des phases de suppression de contrainte et de sélection jusqu'à ce que soit toutes les contraintes ont été éliminées, soit on aboutit à un sous graphe cyclique dans lequel toutes les contraintes ont le niveau de priorité maximal (required).

Lorsque l'algorithme parvient à sélectionner toutes les contraintes, la solution obtenue peut ensuite éventuellement être améliorée en rajoutant progressivement certaines des contraintes qui ont été supprimées : la solution finale est alors du type « locally graph better ».

3.4.8 Indigo

Indigo ([Borning, & Al., 1996]) est un algorithme de propagation locale qui gère les contraintes d'inégalité, toujours dans le contexte des graphes de contraintes acycliques. La motivation initiale de la construction de cet algorithme part du constat que les contraintes d'inégalité ne sont soit pas gérées par les systèmes de contraintes existants, soit simplement vérifiées : l'objectif est donc de mettre en œuvre un algorithme de propagation locale capable de prendre en compte les inégalités dans le contexte des graphes contraintes acycliques.

L'idée générale de l'algorithme consiste à propager non pas des valeurs mais des intervalles désignant les limites inférieures et supérieures des valeurs que peuvent prendre les variables au sein du graphe. Les contraintes de maintien dites « implicites » permettent ensuite de déterminer les valeurs de chacune des variables : la valeur reste inchangée si celle-ci reste incluse dans l'intervalle propagé. Dans le cas contraire, la variable prend la borne de l'intervalle propagé la plus proche de la valeur avant perturbation.

La gestion des hiérarchies de contrainte passe par l'utilisation d'un prédicat de comparaison des solutions comme dans les algorithmes DeltaBlue et SkyBlue. Mais dans le cadre des contraintes d'inégalité, le « locally-predicate-better » est remplacé par une version plus appropriée nommée « locally-error-better ». La notion « d'erreur de satisfaction » est introduite : l'erreur est nulle lorsque la contrainte et satisfaite et augmente lorsque l'on s'éloigne de la solution. Par exemple, pour la contrainte « a =b » l'erreur peut être définie simplement comme | a – b |. Une solution est alors définie comme « locally-error-better » lorsqu'il n'existe pas de « meilleure » solution au sens de ce prédicat. Intuitivement, une solution S_1 est meilleure qu'une autre solution S_2 si il existe un indice k tel que S_1 et S_2 possèdent une erreur totale identique pour les k niveaux les plus importants de la hiérarchie de contrainte et si la S_1 solution donne une erreur totale plus faible que S_2 au niveau k+1. Une hiérarchie donnée peut facilement posséder plusieurs solutions qui soient locally-error-better.

Une raison de préférer le prédicat « locally-error-better » dans le cadre des interfaces utilisateurs est donnée dans [Borning, & Al., 1996] : celle-ci est liée à l'utilisation de la souris et à la fréquence d'échantillonnage de sa position. Si l'utilisateur déplaçait la souris infiniment lentement, les deux comparateurs locally-predicate-better et locally-error-better seraient équivalents : en présence de contraintes de limites, un objet pourrait être déplacé graphiquement progressivement jusque sa limite puis s'arrêterait. En raison de l'échantillonnage temporel des événements de la souris, si l'utilisateur déplace celle-ci rapidement, les positions intermédiaires sont ignorées par le système : l'objet graphique serait « appelé » directement au-delà de sa limite. Dans ce cas, les comparateurs répondent différemment. Le premier, locally-predicate-better ne permet de rien faire : l'action de l'utilisateur ne peut être prise en compte et l'objet graphique n'est pas déplacé. En revanche, en utilisant le comparateur locally-error-better, il est possible de générer la solution qui déplace l'objet jusqu'à la limite autorisée et propose donc un comportement qui semble plus adapté au cas des interfaces graphiques.

3.4.9 Purple et Deep-Purple

Purple est un Solver de contraintes gérant les systèmes d'équations linéaires et admettant les cycles dans le graphe de contraintes. Il est inspiré de l'algorithme utilisé dans CLP(R) ([Jaffar & Al, 1992]) avec en plus la possiblité de gérer les hiérarchies de contraintes (plus précisément, la

gestion des contraintes « required » et « preferential ». Les variables dans Purple sont de trois formes possibles : « parametric », « non-parametric » et « known ». Les variables parametric peuvent prendre arbitrairement n'importe quelle valeur. Les variables nonparametric, en revanche sont définies par des expressions linéaires portant sur des variables parametric et des constantes. Finalement, les variables dites « known » sont des cas particuliers de variables nonparametric, dont les valeurs sont définies par des constantes.

La méthode principale de Purple est appellée pour résoudre un niveau de la hiérarchie : à ce stade, les contraintes sont transformées en leur forme normale linéaire lorsque c'est possible ou rangées dans un liste d'attente « delayed constraints » dans le cas contraire. Par exemple pour la contrainte a * b = c, la forme normale linéaire est possible si l'une des deux variables a ou b est marquée comme « known ». Les variables de la contrainte traitée marquées comme nonparametric et known sont alors remplacées par leurs valeurs de définition, puis la contrainte est simplifiée et le résultat mis sous forme normale : si le résultat est une contradiction, la contrainte est ignorée, sinon, la contrainte est conservée et l'information supplémentaire qu'elle apporte (une variable parametric peut être transformée en nonparametric en lui attribuant une valeur) est propagée.

Deep-Purple, en revanche est un Solver partiel (incomplet) de problèmes de contraintes dans le cadre de contraintes linéaires d'inégalités acceptant les cycles dans le graphe de contraintes : le problème auquel s'attaque Deep-Purple est complexe et pour être efficace deux différentes techniques sont utilisées consécutivement afin de chercher une solution.

La première technique consiste à réécrire toutes les contraintes d'inégalités sous la forme d'une paire de contraintes : une équation linéaire n-aire d'une part, et une inéquation unaire d'autre part. L'équation linéaire prend alors la priorité maximale (required) tandis que l'inéquation conserve le niveau de priorité de la contrainte d'inégalité original. Par exemple, la contrainte « a >= b » est transformée en « a – b = u » d'une part, avec le niveau de priorité maximal, et en « u = 0 » d'autre part avec le niveau de priorité initial de la contrainte. Le système d'équations linéaires est alors confié à l'algorithme Purple, qui réduit et simplifie le système, puis à indigo qui tente de trouver une solution lorsque le graphe restant n'est pas cyclique.

Lorsque le graphe résultant est cyclique et que la première technique échoue, Deep-Purple utilise une deuxième technique : celle-ci, fondée sur le même principe que la première technique, fait également appel à Purple pour gérer les contraintes d'inégalité, mais consiste à resserrer les intervalles de variations possibles imposés par les contraintes. Des cas particuliers sont employés suivant la nature de chaque contrainte, différenciant les contraintes d'inégalité unaires, les équations linéaires et les inégalités binaires. A chaque fois qu'une variable reçoit une valeur, l'algorithme vérifie que la nouvelle valeur est toujours compatible avec les limites admissibles, générant éventuellement des exceptions du type « constraints_too_difficult », ce qui abouti à un échec.

Deep-Purple est incomplet, mais assure tout de même que lorsque une solution est trouvée, celle-ci a la propriété locally-error-better. Il reste tout de même un algorithme difficilement utilisable sauf dans les cas ou le graphe peut être transformé en problème acyclique (et ensuite confié par exemple à l'algorithme Indigo) et est le goulot d'étranglement d'Ultraviolet qui sera présenté dans la section suivante de ce document.

La littérature (disponible) sur ces deux algorithmes n'abonde pas, et l'essentiel de leur

description se trouve dans [Borning & Freeman-Benson, 1998].

3.4.10 Ultraviolet

Les motivations propres au développement de l'algorithme Ultraviolet ([Borning & Freeman-Benson, 1995]) proviennent de deux limitations des algorithmes de propagation locale existants : l'impossibilité de gérer les cycles d'une part, et celle de prendre en compte les contraintes non fonctionnelles d'autre part. Si la plupart des problèmes d'interface utilisateur peuvent être entièrement définis au moyen de graphes de contraintes acycliques ([Borning & Freeman-Benson, 1998]), il semble néanmoins trop fastidieux et contraignant de demander à l'auteur ou au programmeur d'être constamment à l'affût afin d'éviter de construire des cycles. Par ailleurs, les contraintes non fonctionnelles semblent particulièrement importantes dans le cadre des interfaces utilisateurs : en particulier les contraintes d'inégalités qui permettent d'établir des limites aux valeurs des variables sont inévitables.

Ultraviolet veut répondre à ces deux manquements : il gère les cycles dans le graphe de contraintes dans le cadre des contraintes d'égalité et inégalités linéaires sans faire d'hypothèse préalable sur les domaines des variables. Ultraviolet n'est pas directement un algorithme de résolution de contraintes : c'est un algorithme de plus large envergure (méta algorithme) qui consiste à partitionner le graphe de contraintes en différentes régions « homogènes » pour ensuite appliquer localement à chacune de ces régions un sous-solver adapté au type de contraintes qu'elles contiennent. Il regroupe ainsi différents algorithmes observés précédemment dans un contexte plus global :

- deux solvers de propagation locale : Blue pour les contraintes fonctionnelles et Indigo pour les inégalités et autres contraintes numériques

- deux solvers pour gérer les cycles dans les graphes : Purple pour les cycles dans le contexte de contraintes de systèmes linéaires, et Deep Purple pour les cycles dans le contexte des contraintes linéaires d'inégalité. Comme nous l'avons remarqué dans la section précédente (3.4.9), Deep-Purple est le tendon d'Achille d'Ultraviolet. Leurs auteurs envisagent de le remplacer par une version adaptée de QOCA (section 3.4.5)

Ultraviolet introduit la notion de « variables partagées » : ces variables sont mitoyennes entre différentes région du graphe et sont utilisées par les différents algorithmes pour communiquer, échanger des valeurs de perturbation ou de propagation. Les données échangées entre différentes régions sont d'une part les valeurs de ces variables partagées et d'autre part les « walkabout strength » qui décrivent le degré de priorité de la contrainte la plus faible qu'il faudrait désélectionner pour pouvoir assigner n'importe quelle valeur à cette variable : ces valeurs sont utilisées pour la gestion des hiérarchies de contraintes.

Par ailleurs, par souci d'efficacité, Ultraviolet intègre l'idée de compilation : il est courant dans les interfaces graphiques utilisateur, que des perturbation de nature identiques se répètent. Par exemple, lorsque l'utilisateur déplace un objet graphique au moyen de la souris, l'action de perturbation des coordonnées graphiques de l'objet se répète jusqu'à la fin du mouvement. Dans un tel cas, il n'est pas nécessaire à chaque perturbation similaire (même variable perturbée mais avec une valeur différente) de répéter l'étape de planification : au contraire, le plan d'exécution permettant de résoudre le graphe à partir d'une perturbation sera « compilé » et mémorisé de sorte à pour être reexécuté rapidement. Pour ce faire, indigo introduit la notion

de « contrainte d'édition » (edit constraint) qui énonce simplement une propriété du type « x = c » où x est une variable du système et c une valeur arbitraire qui n'est pas connue au moment de la compilation et correspondra typiquement aux actions de l'utilisateur. La phase de compilation consiste donc à mettre en évidence un moyen de résoudre le graphe de contraintes lorsque la valeur de c est modifiée et à transformer ce « moyen » en un bloc programme qui pourra être appelé efficacement. Il est bien entendu généralement nécessaire de pré-compiler un ensemble important de plans, correspondant d'une part à différentes contraintes d'édition (une contrainte d'édition devra être ajoutée pour chaque variable sur laquelle l'utilisateur a la possibilité d'agir) et à différents ensembles de contraintes afin de simuler une certaine forme d'incrémentalité du système.

3.5 Conclusion

Les différents types de contraintes, les propriétés des relations qu'elles établissent, le nombre de variables de sorties, les moyens dont elles disposent pour aboutir à la solution génèrent autant de cas de figure et multiplie le nombre d'algorithmes proposé pour la résolution de problèmes. Ces algorithmes sont parfois également orientés vers des types d'application déterminés servant de base à l'établissement d'un compromis entre les performances, la souplesse, l'expressivité et la rapidité. Ils varient donc de cas relativement simples, efficaces mais traitant des problèmes très spécifiques à des cas beaucoup plus complexe, s'intéressant à des problèmes plus généraux et forcément moins rapides.

Une synthèse relativement exhaustive sur les algorithmes de propagation locale est disponible par exemple dans la thèse de Gilles Trombetoni ([Trombettoni, 1997b]). En particulier, celle-ci propose une classification des différents algorithmes en fonctions du type de contraintes traitées, et de conditions d'utilisation, et aborde de manière détaillée le problème de la complexité de ces algorithmes. Nous en retenons essentiellement que le problème général de la propagation locale, envisageant des graphes cycliques de contraintes non nécessairement linéaires et non nécessairement fonctionnelles est un problème NP-complet difficile à implémenter dans le cadre d'applications temps-réel.

Dans notre projet, MusicSpace, les contraintes portent à priori sur les positions de sources sonores par rapport à celle d'un auditeur. Les variables manipulées sont donc des nombres flottants qui représentent des coordonnées et varient continûment. Le cadre qui nous concerne paraît être le cadre le plus général, combinant les contraintes non fonctionnelles (contraintes de limite et d'inégalité), les contraintes multi-output, les graphes multi-way, et des relations non nécessairement linéaires. Par ailleurs il est nécessaire également de tenir compte de cycles dans le graphe de contraintes.

Cet ensemble de « mauvaises » propriétés penche a priori en faveur d'un algorithme relativement complexe, proche d'Ultraviolet (voir section 3.4.10 de ce document) par exemple, le plus complet, combinant un méta algorithme capable de décomposer le graphe de contraintes en sous-parties et des sous-solvers spécifiques qu'il emploie en fonction de la nature de chaque sous-graphe.

En revanche, nous avons dans le cadre de MusicSpace une propriété originale des contraintes qui n'est pas souvent exploitée dans la littérature : manipulant des positions de sources sonores, il est impératif pour le système de spatialisation sous-jacent (pour qui les positions

relatives des sources seront interprétées en commande de spatialisation) que ces sources se déplacent de manière relativement continue. En effet, le saut brutal d'une source sonore d'une position à une autre produirait sur le résultat des modifications de la scène sonore a priori incohérente ainsi que probablement une dégradation du signal résultant par des clics par exemple. Il en résulte une certaine notion de continuité également pour les contraintes et l'algorithme de résolution spécifiant intuitivement qu'une perturbation faible du système induit une propagation faible vers les variables.

Nous avons choisi, pour notre projet MusicSpace, de construire un algorithme ad hoc largement inspiré des algorithmes existants, et en particulier de SkyBlue. Cet algorithme gère simplement les cycles en les vérifiant, tient compte des contraintes multi-way, multi-output, et des contraintes non fonctionnelles. En revanche la notion de hiérarchie de contraintes n'a pas été abordée dans sa version actuelle et toutes les contraintes sont donc « requises ». La deuxième partie de ce document, consacrée à MusicSpace dresse un inventaire des contraintes utilisées, puis détaille l'algorithme correspondant.

Conclusion de la première partie

Comme nous l'avons mentionné, la spatialisation du son et la gestion de l'espace comme paramètre de jeu compositionnel figure depuis longtemps au cœur des préoccupations des compositeurs, au point d'engendrer parfois des réalisations et productions de dimensions techniquement colossales.

Les systèmes de spatialisation sont venus au secours des compositeurs, en réduisant considérablement les moyens techniques à mettre en œuvre pour tenir un discours musical qui s'appuie sur le paramètre spatial : un nombre important de techniques ont été présentées

De plus, des travaux récents d'interfaçage de ces systèmes ont permis d'unifier les paramètres de contrôle et de rendre la technique sous-jacente la plus transparente possible à l'utilisateur.

Pourtant ces systèmes sont le plus souvent sous-exploités, en particulier en raison du manque d'interfaces de contrôle sémantiquement cohérentes, qui fassent sens et soient adaptées aux nécessités des utilisateurs : le paramètres proposés sont ainsi non seulement peu intelligibles pour les utilisateurs mais surtout il ne permettent pas d'établir et de maintenir les relations qui existent entre les différentes sources sonore d'une scène musical, et qui sont nécessaires à garantir la cohérence du résultat sonore.

Nous avons finalement, dans cette première partie, présenté la programmation par contraintes, paradigme permettant typiquement d'exprimer des relations arbitraires portant sur les variables d'un système et produisant des méthodes de calcul permettant de calculer des valeurs à ces variables, de sorte que les relations énoncées soient vérifiées. Cette étude des algorithmes de moteur de résolution de contraintes existant a laissé de coté les algorithmes dits de satisfaction pour s'intéresser davantage aux algorithmes dits de propagation, largement étudiés dans le cadre d'interfaces utilisateurs, et connus pour leurs capacités à produire des systèmes réactifs.

C'est la rencontre de ces différents domaines que nous proposons dans la partie suivante de ce document, où nous décrivons MusicSpace, une interface de contrôle de la spatialisation tirant partie de la programmation par contraintes pour exprimer les relations existantes entre les différentes sources d'une scène sonore, et garantir la cohérence du résultat sonore.

Deuxième Partie
MusicSpace

Notre objectif est de construire une interface de contrôle de scènes sonores permettant d'exploiter au mieux les outils de spatialisation existants, en tenant compte en particulier des relations qui existent entre les différentes sources sonores.

Nous souhaitons pour cela reprendre les points positifs des interfaces existantes. En particulier, nous nous attachons à la représentation intuitive sous la forme d'une schématisation bidimensionnelle de la scène sonore à l'image d'une vue d'oiseau – comme dans l'interface circle, par exemple (voir Figure 14) – qui, à l'instar de l'alignement curseurs de niveau sur la console de mixage, permet à l'utilisateur se se faire une idée globale et synthétique des rapports existant entre les paramètres principaux de la spatialisation : la distance et l'azimut des sources sonores par rapport au point d'écoute. Nous ajoutons également la représentation du point d'écoute ainsi que la possibilité de le déplacer. Ce type de manipulation, souvent absent des interfaces de contrôle exsitantes, permet pourtant d'effectuer de modifications intéressantes sur le rendu sonore, beaucoup plus facilement que s'il avait été question de déplacer toutes les sources simultanément.

A ce système de base nous intégrons un système permettant de décrire, représenter et maintenir les différentes relations qui existent entre les sources sonores. Nous l'avons vu, et nous le décrirons encore au sein d'un exemple précis, ces relations sont fondamentales pour garantir la cohérence du résultat sonore. Nous nous intéressons particulièrement à la programmation par contrainte, paradygme qui nous semble le plus adapté à la représentation et au maintient de ces relations. Plus précisément nous feront usage des techniques de propagation et décrirons au Chapitre 5 les contraintes mises en œuvre d'une part et l'algorithme construit d'autre part.

Au dela du cahier des charges présenté, nous mettons l'accent sur la simplicité d'utilisation du système : à ce titre, nous reprenons les fonctionnalités standards des interfaces graphiques, en particulier au niveau de la manipulation à l'aide de la souris, des modes de sélection des objets, de sélection multiple et d'édition. Mais nous insistons également sur la simplicité avec laquelle l'auteur pourra construire le jeu de contraintes pour une scène sonore donnée : cette construction s'effectue de façon incrémentale et entièrement graphique. A tout moment

l'utilisateur peut tester l'ensemble de contraintes qu'il a défini et éventuellement le modifier.

Pour finir, notre système présentera deux modes de travail, auteur d'une part, et utilisateur d'autre part. En mode auteur, l'intégralité de la scène, comprenant les sources sonores, les contraintes et autres éventuels objets graphiques est représentée et modifiable. En mode utilisateur, au contraire, seul les éléments utiles utiles sont visibles et seuls les paramètres significatifs sont manipulables.

La section suivante décrit notre système MusicSpace dans sa version de base et met en avant ses principales fonctionnalités. A cette occasion nous présentons un exemple simple et montrons comment même un cas pourtant élémentaire peut conduire à des situations de mixage incohérentes : cette situation nous mène au chapitre suivant qui décrit le système de contraintes ajouté à MusicSpace, tant au niveau de l'algorithme de résolution que de l'inventaire des contraintes utilisées en essayant d'expliciter, pour chacune d'entre elle sa justification musicale. Finalement, nous détaillerons dans cette partie du document les différents systèmes de spatialisation utilisés en précisant, pour chacun d'entre eux, ses particularités et les conséquences de ces particularités sur les contraintes utilisées.

Chapitre 4
Description du système de base

MusicSpace est une interface graphique de contrôle de la spatialisation du son : la scène sonore y est représentée sous la forme de la projection dans un plan horizontal d'un ensemble de sources sonores ainsi que d'un « avatar », point de référence symbolisant l'auditeur. Ce système purement graphique est connecté à un module de spatialisation auquel il communique les informations géométriques de distance et d'angle relatives entre les sources et l'avatar.

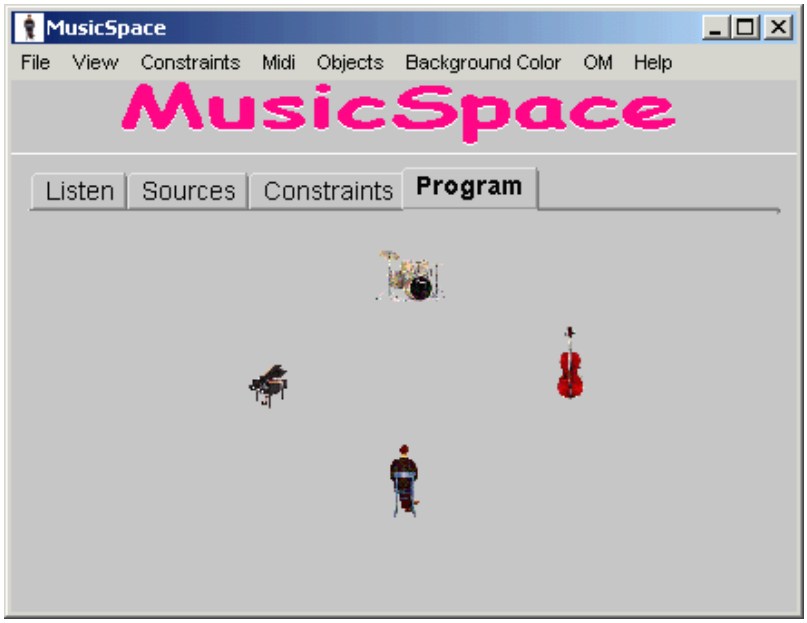


Figure 39 : MusicSpace, description du système de base

La Figure 39 donne un premier aperçu général de notre système. Dans cet exemple, le fichier musical « Trio Jazz » est chargé : l'interface représente d'une part une icône symbolisant l'auditeur (l'avatar) et d'autre part les différentes icônes correspondant à chacune des sources sonores. Il y en a trois dans cet exemple, représentant un piano, une contrebasse et une batterie.

L'utilisateur a la possibilité d'ajuster son point d'écoute en modifiant à l'aide de la souris la position de l'avatar : les modifications de positions relatives entre l'avatar et chacune des sources sont alors immédiatement transmises au système de spatialisation

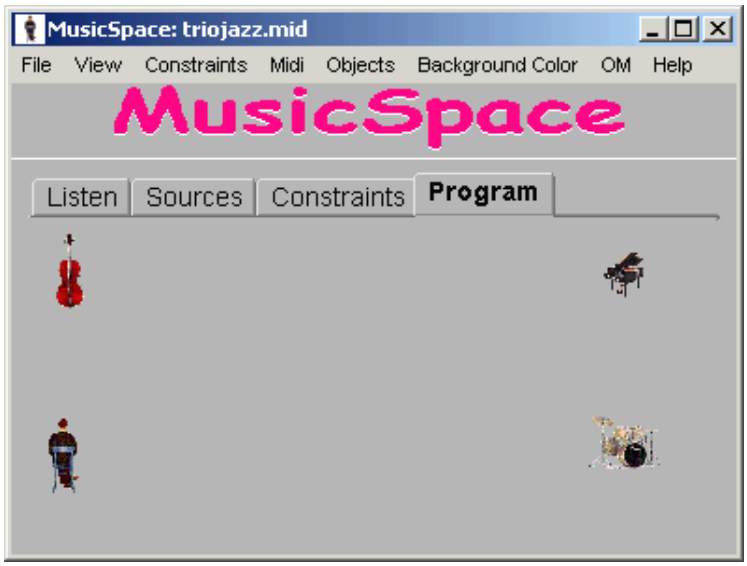


Figure 40 : organisation incohérente (1)

Libre de modifier la position des sources sonores et de l'avatar à sa guise, un utilisateur peut facilement transformer l'organisation du trio de jazz présentée sur la Figure 39 en celle de la Figure 40, qui présente une singularité sur le plan visuel mais n'a aucun intérêt sur le plan sonore. En particulier, le défaut majeur de cette organisation des sources est qu'elles sont trop éloignées les unes des autres pour être entendues simultanément : en explorant la scène en déplaçant l'avatar, chaque instrument est entendu séparément et la notion de trio dans laquelle précisément trois instruments devraient jouer ensembles est détruite.

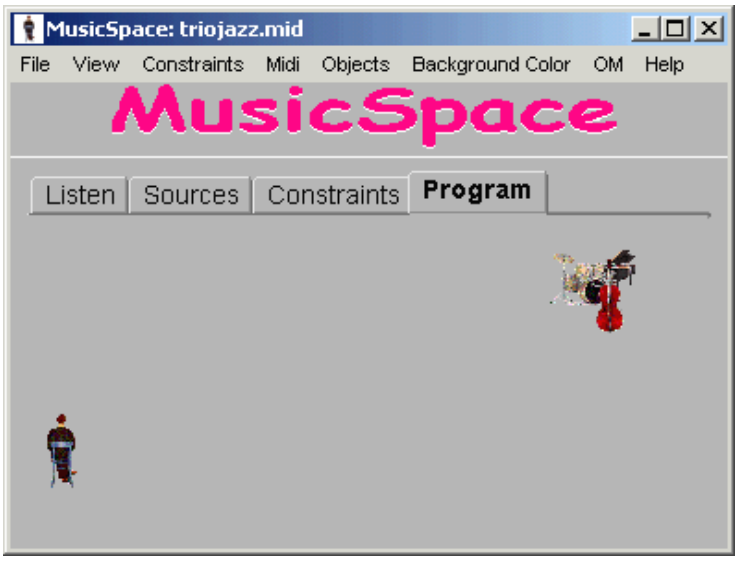


Figure 41 : organisation incohérente (2)

Pour une raison différente, la Figure 41 présente une organisation des sources et de l'avatar qui ne convient pas non plus. On reproche à ce type de situation :

- Ensemble de sources trop ramassé par rapport à l'avatar : l'espace sonore, stéréophonique

par exemple est exploité couramment par les ingénieurs du son pour permettre de distinguer chaque source les unes des autres.

- Image sonore décentrée (vers la droite) : nous sommes habitués à un certain équilibre

Ce système, dans sa version initiale, présente déjà un intérêt non négligeable. Il donne d'une part, par la représentation de l'avatar, une matérialisation du point d'écoute mais en plus il offre la possibilité d'agir sur ce point de vue ce qui n'est généralement pas possible dans les interface de contrôle des système de spatialisation existantes. Déplacer le point d'écoute reviendrai alors à déplacer toutes les sources sonores de la scène simultanément et d'une manière telle que les sources conservent leurs distances les unes aux autres ce qui serait pratiquement infaisable à l'aide de la souris. En revanche notre système amène, comme nous l'avons montré, facilement à des situations de mixage incohérentes, surtout lorsqu'il est manipulé par un utilisateur non-expert en mixage et spatialisation. C'est ce problème que nous nous proposons d'aborder dans la section suivante de ce document.

Chapitre 5
Les contraintes dans MusicSpace

Si nous parvenons, comme dans le cas de l'exemple précédent, à des situations de mixage incohérentes, c'est simplement parce que notre système de base est trop ouvert. Dans la majeure partie des cas, les positions des sources sonores sont pas indépendantes les unes des autres et ne devraient donc pas être modifiées aussi librement. Des relations fortes relient les positions des sources sonore les unes des autres et doivent être respectée pour atteindre un résultat cohérent. C'est ce type de relations que nous proposons d'inclure dans notre système, et de représenter sous forme de contraintes.

Au système de base décrit précédemment vient donc s'ajouter un système de contraintes permettant, comme nous l'avons expliqué dans les sections précédentes, de définir un ensemble de relations qui opèrent sur les positions des sources sonores, éventuellement en fonction de la position de l'avatar.

Les solutions que nous proposons dans MusicSpace s'intéressent non seulement à la technique des contraintes mais aussi aux modalités d'interaction proposées à l'utilisateur pour définir facilement quel va être l'ensemble de contraintes utilisées pour une scène sonore donnée ainsi que la manière dont le graphe de contraintes va être représenté.

Ainsi, les contraintes sont représentées graphiquement, de la même manière que les sources sonores ou l'avatar, de sorte que l'auteur de la scène puisse en avoir un aperçu global à tout instant. Le processus – incrémental – d'ajout ou suppression d'une contrainte est simple : l'utilisateur commence par sélectionner les objets qu'il souhaite contrainte au moyen des méthodes de sélection graphiques standards (sélection multiple à l'aide de la souris en encadrant les objets à sélectionner, click de la souris sur chacun des objets en gardant une touche de maintien de sélection enfoncée…) propres aux interfaces graphiques. L'utilisateur choisit ensuite la contrainte qu'il souhaite poser sur ces objets par l'icône qui la représente au sein de la palette de contraintes telle que représentée en Figure 42.

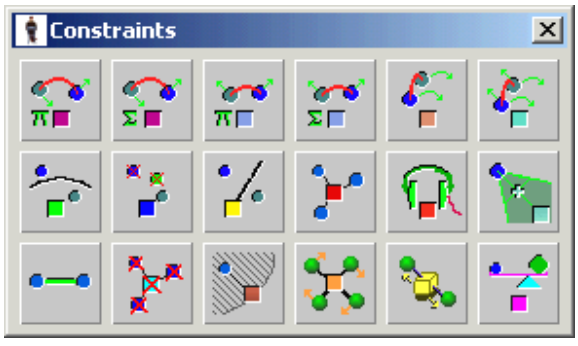


Figure 42 : palette de contraintes

La contrainte est alors représentée graphiquement sous la forme d'un cercle reprenant la couleur de l'icône correspondante et éventuellement un symbole supplémentaire permettant à l'utilisateur de l'identifier facilement des segments permettent de repérer visuellement quelles sont les variables dépendantes de cette contrainte. Par exemple, la Figure 43 représente deux contraintes posées sur différentes variables. La première, est une contrainte n-aire (ici ternaire) posée sur les objets v1, v2, v3. La seconde est une contrainte de limite posée sur un seul objet (v3). Il est à noter que l'avatar, faisant quasiment toujours partie des variables prises en compte dans ces contraintes puisque toutes les distances par exemple sont mesurées par rapport à sa posisition, est implicitement lié au contraintes. Pourtant, par mesure de clarté, ce lien n'est pas représenté graphiquement.

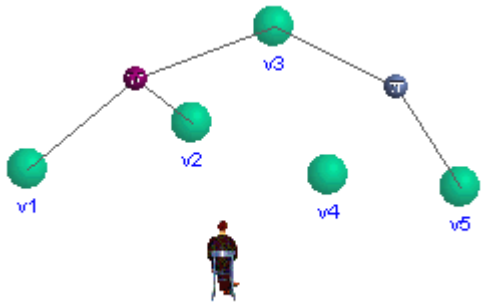


Figure 43 : Deux contraintes et Cinq variables

Un inventaire des contraintes est proposé dans un premier temps, donnant un panorama des relations et des types de relations que l'on souhaite établir entre les sources sonores. Puis, dans un deuxième temps seront abordés les aspects relatifs aux problèmes algorithmiques posés, et à l'implantation informatique choisie. Nous terminerons la description de notre système en détaillant chacun des systèmes de spatialisation étudiés et leurs particularités.

5.1 Inventaire des contraintes

Nous décrivons dans cette section l'ensemble des contraintes qui ont été implémentées dans notre système. Nous nous concentrons sur l'interprétation « musicale » de ces relations qui, portant sur les positions des sources sonores, ne s'expriment a priori qu'en termes géométriques. Nous laissons en revanche pour l'instant de coté les détails d'implémentation des contraintes : ceux-ci interviendront dans la description algorithmique de notre système.

5.1.1 Contraintes de base

5.1.1.1 Rapport des distances 2 à 2 constant

La première contrainte de base veut que le rapport des distances des objets contraints pris deux à deux reste constant. Derrière cette formulation se cache l'idée intuitive que l'équilibre entre un ensemble de sources appartenant à un même groupe doit être maintenu, quelles que soient les modifications apportées à l'une ou l'autre des sources sonores de ce groupe. La Figure 44 donne un exemple de trois objets v_1, v_2 et v_3 liés par une telle contrainte. Si l'utilisateur apporte

une perturbation à v_1, par exemple, en diminuant sa distance à l'avatar de moitié, l'effet de la propagation rapprochera les autres objets v_2 et v_3 d'un rapport de distance identique.

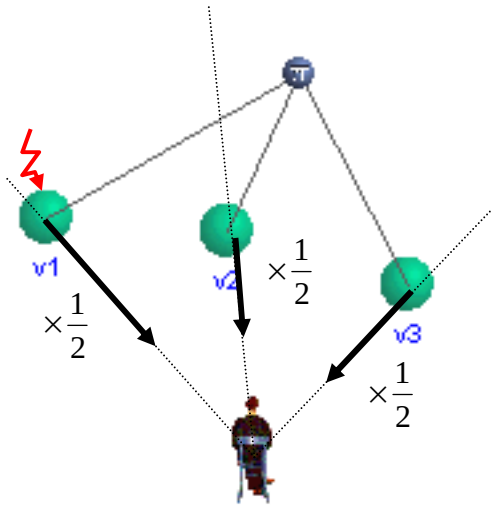


Figure 44 : rapport des distances 2 à 2 constant

Cette contrainte est la seule de notre système qui soit déjà implantée dans les consoles de mixage : elle correspond à la notion de « sous-groupe » sur les consoles les plus anciennes qui permettent d'effectuer le « pré-mixage » d'un ensemble de pistes pour ensuite n'intervenir de manière globale, qu'au niveau de ce mixage intermédiaire, et non sur les pistes elles-mêmes. La modification est ainsi plus simple, et préserve l'équilibre entre les pistes du groupe. Typiquement, en musique, cette fonctionnalité permet de regrouper des pupitres de cordes, ou de cuivres par exemple, ou encore l'ensemble des microphones (pouvant atteindre une dizaine) utilisés pour la prise de son d'une batterie.

Sur les consoles à commande numérique, plus récentes, cette contrainte apparaît sous la forme de « groupe » qui lie – parfois même mécaniquement – les curseurs de volume de l'ensemble des pistes concernées. Lorsque l'utilisateur ajuste le niveau d'une des pistes groupées, cette « contrainte » applique un déplacement similaire (à concurrence de la marge de déplacement des potentiomètres) à chacun des curseurs du même groupe. Sur une échelle logarithmique, la contrainte correspondante correspond à des différences des distances 2 à 2 constantes tel que représenté sur la Figure 45.

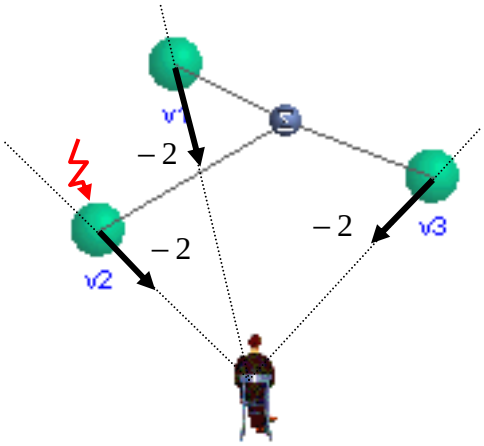


Figure 45 : différence des distances constante

Intuitivement, dans la version sous forme de ratio constant comme dans celle sous forme de différence constante, rapprocher (resp. éloigner) une variable de l'avatar aura pour effet de rapprocher (resp. éloigner) également les autres variables contrainte, d'une quantité dépendant de la perturbation initiale.

5.1.1.2 Produit des distances à l'avatar constant.

La version duale de cette la contrainte précédente est le « produit des distances constant ». Celle-ci, à l'inverse de la contrainte précédente, a pour effet lorsqu'on rapproche (resp. éloigne) une source sonore de l'avatar d'en éloigner (resp. rapprocher) les autres sources sonores contraintes de sorte à ce que le niveau total soit constant.

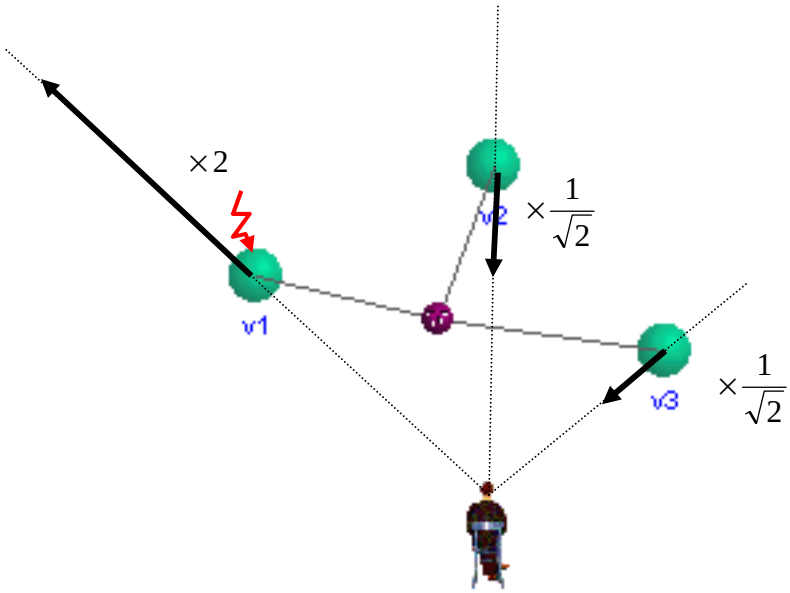


Figure 46: produit des distances constant

À l'instar de la contrainte précédente, celle-ci possède également sa version linéaire : « somme des distances constantes », comme montré sur la Figure 47.

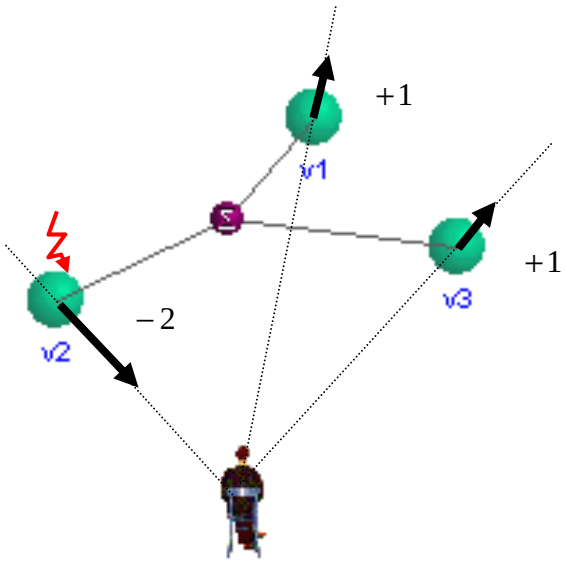


Figure 47 : somme des distances constante

5.1.1.3 Différence angulaire constante

Ces relations exprimées sur les distances des sources sonores à l'avatar trouvent également des interprétations intéressantes au niveau angulaire, tel que par exemple l'angle sous lequel l'avatar voit ces sources. La contrainte « différence angulaire constante » est une contrainte n-aire (n >= 2) qui vise à conserver l'écart angulaire sous lequel l'avatar voit les sources deux à deux.

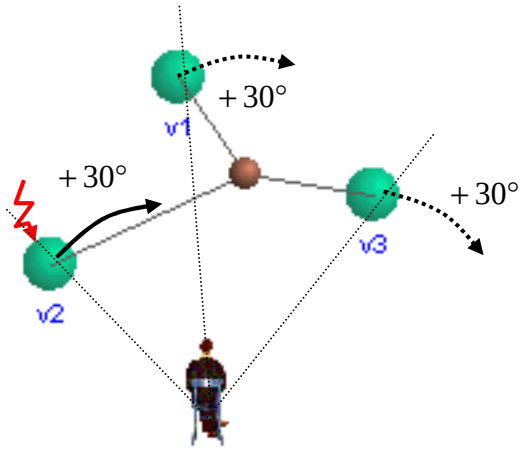


Figure 48 : écarts angulaires constants

Ainsi, comme représenté sur la Figure 48, une perturbation angulaire apportée a v2 par exemple est propagée aux autres variables (en pointillés) de telle sorte que l'angle (v1, avatar, v2) reste constant. La gestion de l'espace est un aspect fondamental du mixage puisque lorsque deux sources se « chevauchent » autant sur le plan temporel que sur les aspects spectraux, l'organisation de l'image spatiale (stéréophonique par exemple) est un des facteurs sur lequel l'ingénieur du son peut intervenir afin d'éviter des phénomènes de masquage par exemple, et conserver la clarté du résultat sonore.

5.1.1.4 Contrainte de translation

Tandis que les contraintes décrites précédemment ne s'adressaient à la fois qu'à un seul degré de liberté des sources sonores (la distance, ou la position angulaire par rapport à l'avatar), la contrainte de translation entraine une modifications deux variables X et Y de position des sources. La relation exprimée par cette contrainte est que l'écart, en coordonnées cartésiennes entre les objets contraints pris deux à deux doit rester constant. En d'autres termes, la perturbation de positions (en coordonnées cartésiennes) arrivant à l'un des objets contraints doit être propagées de manière identique aux autres objets contraints. Une manière d'exprimer cette relation peut être : pour chaque paire d'objets contraints, la différence des X = cste et la différence des Y = cste.

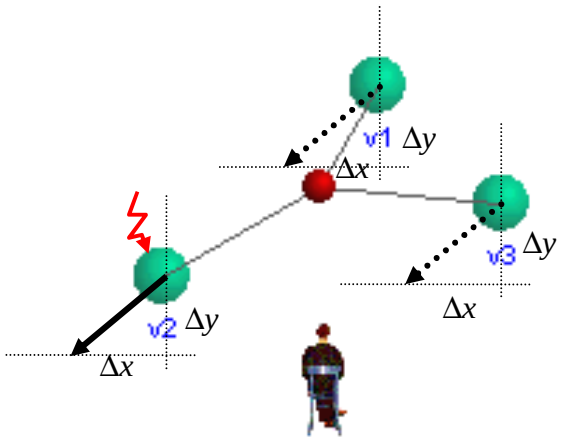


Figure 49 : contrainte de translation

Dans l'exempe représenté en Figure 49, l'objet v2 est déplacé d'une quantité Dx sur l'axe horizontal et Dy sur l'axe vertical. Le résultat de la propagation est un déplacement similaire pour les autres objets contraints, v1 et v3.

Concrêtement, cette contrainte pose le problème de la double représentation des objets en coordonnées polaires et cartésiènes. Dans notre implémentation, la perturbation d'un objet est mesurée en coordonnées polaires : pour réaliser cette contrainte, la perturbation est donc convertie en coordonnées cartésienne, puis propagée vers les autres objets puis reconvertie en coordonnées polaires pour que cette propagation puisse être prise en compte.

Cette contrainte ne possède pas d'interprétation directe au niveau sonore. Elle permet toutefois

de symboliser et représenter par exemple un objet solide émettant des signaux sonores en différents points de sa structure. Par ailleurs, elle trouve également de l'intérêt lorsqu'utilisée en combinaison avec la contrainte de proximité (voir section 5.1.3.1) pour définir par exemple un périmètre constant autour de l'avatar.

5.1.2 Contraintes de limite

Les contraintes de limite représentent les inégalités du système : elle auront pour utilisation principale la détermination des plages admissibles de variations d'un paramètre, typiquement en limitant l'ensemble des positions acceptables pour les objets sonores dans la scène.

Ces contraintes de limite, radiale ou angulaire, ont pour particularité d'exprimer par leur position géométrique la valeur de la limite elle-même : la position de l'objet qui la représente entre donc dans le paramètrage de la contrainte. En contraignant l'objet contrainte lui-même, il est possible de faire entrer ces relations de limite au sein de notions plus complexe telle que part exemple un « ambitus de distances constant ».

5.1.2.1 Limite Angulaire

La contrainte de limite angulaire définit une demi-droite partant de l'avatar et passant par l'objet "contrainte" lui-même. Elle stipule que les objets contraints ne doivent pas franchir cette demi-droite quelle que soit leur position initiale d'un coté ou de l'autre de la demi-droite.

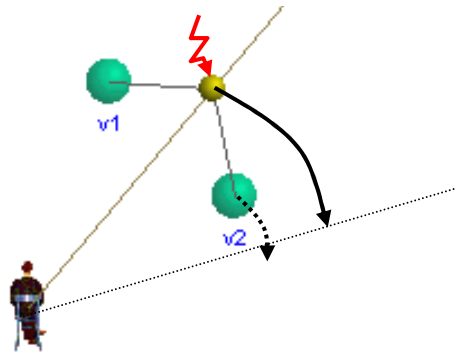


Figure 50 : contrainte de limite angulaire

Plusieurs options sont disponibles pour cette contrainte et permettent de définir si :

- La contrainte peut déplacer un objet contraint.
- Les objets contraints peuvent déplacer l'objet "contrainte" lui-même.

Lorsque aucune de ces options n'est choisie, toute perturbation violant l'inégalité est refusée, qu'il s'agisse d'une perturbation intervenant sur les objets contraints ou sur l'objet « contrainte » lui même.

Lorsque l'option « la contrainte peut déplacer un objet contraint » est activée, une perturbation sur l'objet contrainte est propagée vers les objets contraints pour tenter de rétablir l'inégalité à vérifier : l'objet contraint est alors perturbé en angle uniquement de manière à maintenir sa position du coté de la demi droite ou il se trouvait initialement. Inversement, lorsque l'option « les objets contraints peuvent déplacer l'objet "contrainte"» est choisie, une perturbation apportée à un objet contraint peut être propagée vers la contrainte elle même puis si nécessaire vers les autres objets contraints de manière à maintenir l'inégalité : de la même manière que dans le cas précédent, c'est cette fois ci l'objet « contrainte » qui serait déplacé de manière angulaire.

Dans l'exemple représenté en Figure 51, l'objet contrainte est déplacé vers la droite : L'objet v1, situé initialement à gauche de la contrainte, reste immobile puisque la perturbation ne dérange pas l'inégalité. En revanche, l'objet v2 subit une perturbation angulaire de manière à rester du coté droit de la limite.

Cette contrainte peut être utilisée typiquement pour partionner l'espace auditif en sections angulaires de sorte que les sources sonores restent dans des régions distinctes : en effet la séparation spatiale des sources peut être un moyen efficace pour conserver l'intelligibilité du contenu lorsque plusieurs sources de nature similaire ou occupant la même bande de fréquence ont tendance à se masquer mutuellement.

5.1.2.2 Limite Radiale

La contrainte de limite radiale fonctionne de manière quasi identique à la limite angulaire : la contrainte définit cette fois ci un périmètre circulaire centré sur l'avatar et de rayon égal à la distance entre l'avatar et l'objet « contrainte », que les objets contraints ne peuvent franchir.

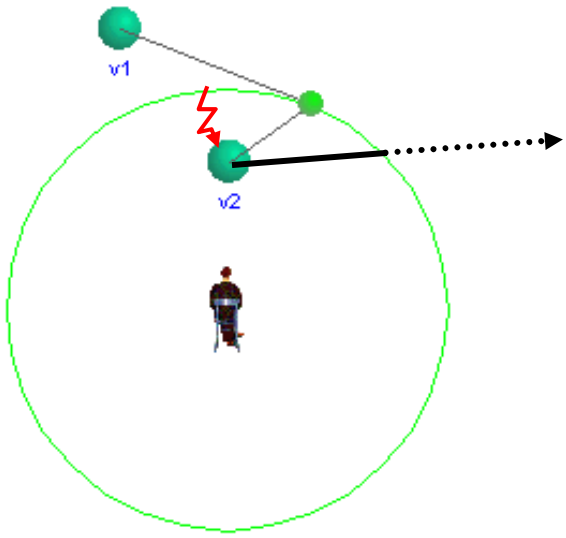


Figure 51 : contrainte de limite radiale

La Figure 51 représente deux objets V_1 et V_2 contraints par une contrainte de limite radiale. Lorsque l'utilisateur déplace l'objet V_2, par exemple, le mouvement est accepté jusqu'à ce qu'il franchisse la limite imposée par la contrainte de limite radiale. La perturbation est ensuite

refusée et l'objet V_2 cesse de suivre le mouvement de la souris.

Les contraintes de limite angulaire et radiale disposent d'une option supplémentaire permettant de « suivre l'avatar » et ainsi de déterminer comment l'inégalité évolue lorsque l'avatar est déplacé. Lorsque cette option est choisie, l'inégalité est à « rayon constant » par exemple, pour la limite radiale. Les déplacements de l'avatar sont propagés comme des perturbations vers la contrainte qui ajuste sa position pour rester à une distance constante de l'avatar et éventuellement propage cette perturbation aux objets contraints pour maintenir l'inégalité. De même, si cette option est choisie dans le cas de la contrainte de limite angulaire, celle-ci ajustera sa position en fonction des déplacements de l'avatar

5.1.3 Contraintes de "mute"

Toutes les contraintes présentées jusqu'à présent portaient uniquement sur les variables de position des objets. Nous montrons ici qu'il est possible d'agir sur d'autres variables (ou propriétés des objets) telles que la variable de « mute » par exemple. Celle-ci, à valeurs booléennes, définit la propriété « muté » ou « dé-muté » qui dans le cas d'une source sonore s'interprète respectivement comme audible ou inaudible. Cette notion de « mute » provient des consoles de mixages où « muter » une voix consiste à l'éteindre complètement en sortie de manière à éviter de générer inutilement du bruit de fond. Nous l'étendons ici à l'ensemble des objets de notre projet en leur donnant une interprétation propre en fonction de leur nature. Pour une contrainte, par exemple, l'état « muté » est l'état dans lequel la relation qu'elle représente entre ses objets contraints n'a plus à être vérifiée. C'est donc par ce mécanisme qu'il va être possible d'activer ou désactiver dynamiquement une contrainte : un exemple d'application en est donné au paragraphe 5.1.3.3.

5.1.3.1 Contrainte de proximité

La contrainte de proximité est une contrainte unaire. Elle définit un périmètre autour de l'objet contraint auquel elle transmet des valeurs de "mute" suivant l'appartenance ou non de l'avatar au périmètre en question. La Figure 52 représente un exemple de cette contrainte dans laquelle trois variables v1, v2, v3 sont attachées à des contraintes de proximité. L'avatar est situé dans les périmètres concernant v1 et v2, mais pas dans celui de v3 : v3 est donc "muté".

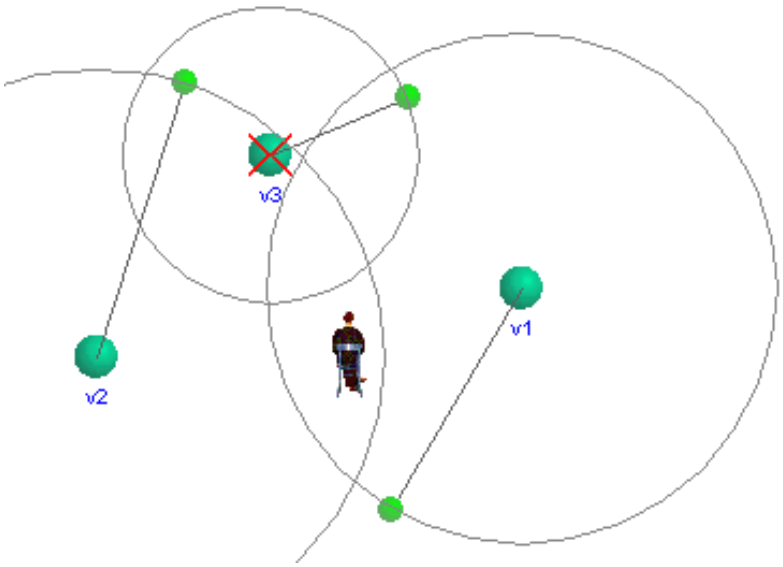


Figure 52 : Contrainte de proximité

Ce type de contrainte permet par exemple de décider dans quelle région de l'espace une source sonore va être active ou inactive : ce type de mécanisme peut être très utile lorsque l'on souhaite créer des scènes sonores complexes comprenant un nombre de sources supérieur au maximum que la machine est capable de gérer, mais qui ne sont jamais toutes jouées simultanément. Les commandes de mute de MusicSpace s'accompagnent alors d'une gestion des ressources de la machine, en libérant des ressources lorsque l'avatar sort d'une zone de proximité pour les affecter à d'autres sources qui doivent être jouées.

5.1.3.2 Groupe de Mute :

La contrainte "groupe de mute" est sensible aux perturbations de type "mute" ou "unmute". Elle propage celles-ci à l'ensemble de ses objets contraints. Elle permet de cette façon, de transmettre simultanément une valeur de mute à un ensemble d'objets et s'apparente directement à la fonctionnalité correspondante dans les consoles de mixages, utilisée généralement pour fermer un ensemble de pistes inutilisée afin d'éviter au maximum d'ajouter du bruit au mixage général.

Cette contrainte se combine relativement bien à la contrainte de proximité décrite précédemment, comme le montre la Figure 53 où, sur un ensemble de 5 variables v1 à v5, deux contraintes de type "groupe de mute" on été définies dans un premier temps et ensuite assujeties chacune à des contraintes de proximité.

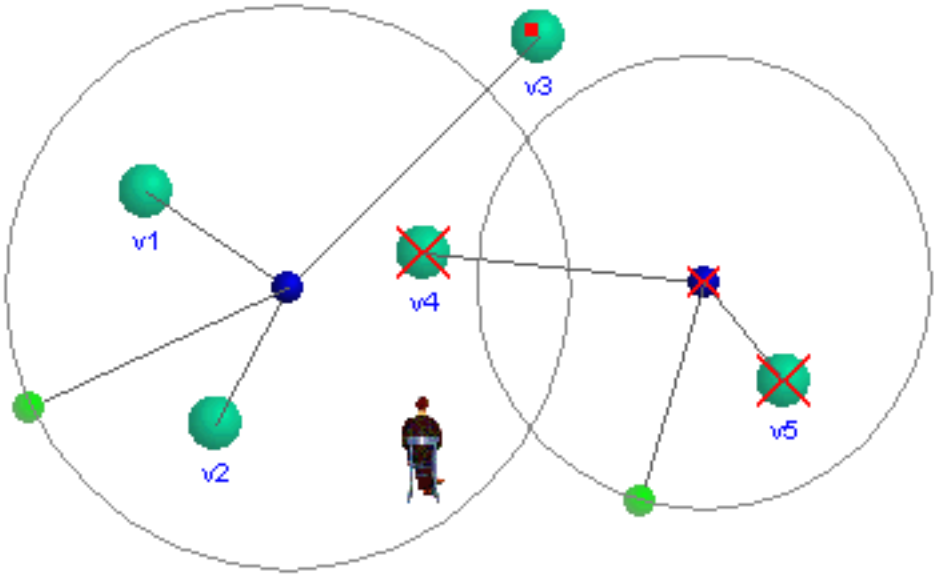


Figure 53 : groupe de mute (1)

Dans cet exemple, les contraintes de proximité définissent deux périmètres centrés sur les contraintes de "groupe de mute" correspondantes. Deux zones sont ainsi définies dans la scène et déterminent les valeurs de mute de v1, v2 , v3 d'une part et de v4, et v5 d'autre part. Ici v1, v2, et v3 sont non mutées tandis que v4 et v5 sont mutées.

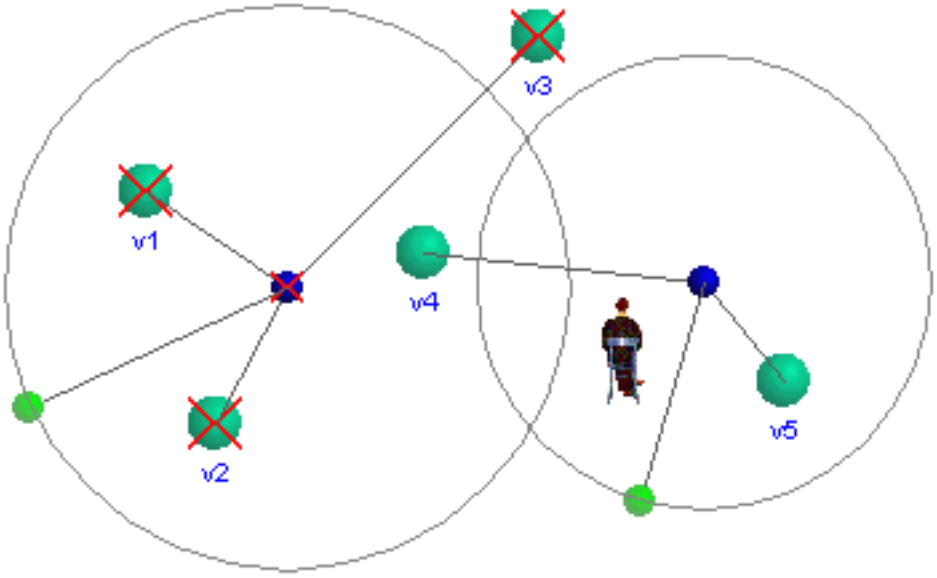


Figure 54 : groupe de mute (2)

En déplaçant l'avatar vers la droite, dans l'autre zone, comme décrit sur la Figure 54, les valeurs

de mute s'inversent : les variables v1, v2 et v3 deviennent mutées, tandis que v4 et v5 repassent en mode "non muté". Dans la zone commune aux deux périmètres, au centre de la figure, toutes les variables seraient non-mutées, tandis qu'à l'extérieur des deux cercles toutes seraient mutées.

Il est intéressant de remarquer dans cet exemple que l'état muté ou non-muté des variables v1 à v5 est découplé de leur position géométrique et fait usage de la position de la contrainte "groupe de mute" comme d'une variable intermédiaire du système.

5.1.3.3 Contrainte : « la plus proche seulement »

Le dernier exemple de contrainte agissant sur les valeurs de mute des variables fait encore un lien entre les valeurs de positions et les valeurs de mute des objets contraints. Dans ce cas, la contrainte propage des valeurs « mute » à tous les objets contraints sauf celui qui se trouve le plus près de l'avatar qui reçoit la valeur non-mute.

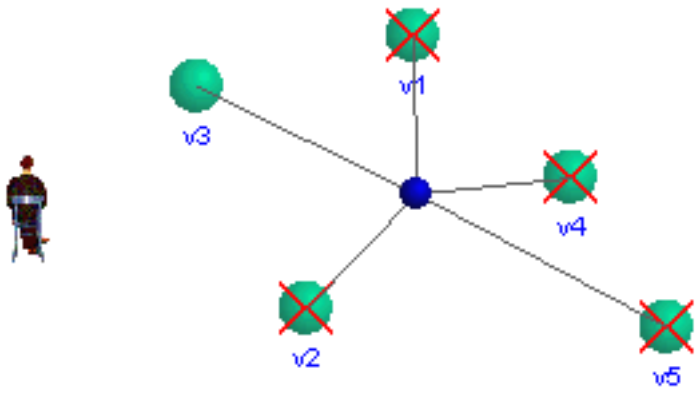


Figure 55 : contrainte : "la plus proche seulement" sur un ensemble de 5 variables

Comme le montre la Figure 55, c'est l'objet v3 qui se trouve le plus proche de l'avatar et donc n'est pas muté. Il est à noter que pour cette contrainte comme pour la contrainte de proximité, le paramètre « mute » est toujours utilisé comme une variable de sortie.

Nous donnons ici (Figure 56, Figure 57, Figure 58) un exemple d'utilisation de cette contrainte mettant en évidence également la possibilité d'activer ou désactiver dynamiquement une contrainte.

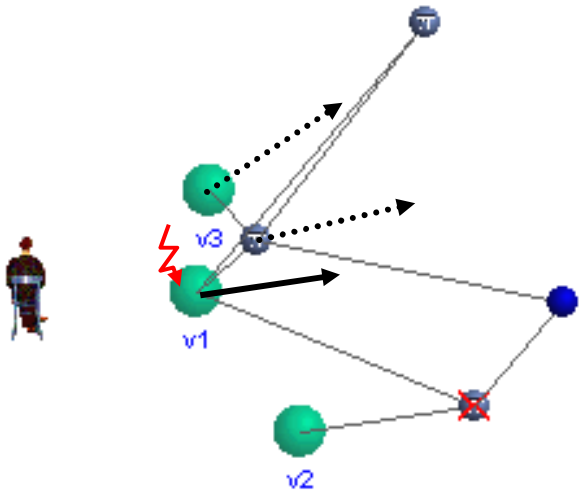


Figure 56 : exemple d'utilisation de la contrainte "le plus proche seulement" (1/3)

L'exemple présenté en Figure 56 décrit trois variables v1, v2 et v3 : v1 et v2 d'une part et v1 et v3 d'autre part sont reliées par des contraintes « ratio des distances constant ». Un déplacement radial de v1 entraînerait donc normalement un déplacement radial d'un ratio équivalent des variables v2 et v3. Mais les deux contraintes « ratio des distances constant » sont elles-mêmes reliées par une contrainte de type « le plus proche seulement » de telle sorte que seule l'une des deux ne peut être active à la fois. Pour finir, la contrainte « ratio des distances constant » reliant v1 et v3 est elle même reliée à v1 par une contrainte du même type. L'objet contrainte lui-même, reliant v1 à v3 réagit aux déplacement de v1 : son activité est remise en cause lorsque sa distance à l'avatar est plus grande que celle de la contrainte reliant v1 à v2.

Ainsi (Figure 56), lorsque v1 est proche de l'avatar, c'est l'objet v3 qui réagit à son déplacement. A mesure que v1 s'écarte de l'avatar, la contrainte reliant v1 à v3 s'écarte également jusqu'au moment ou sa distance à l'avatar devient supérieur à celle de la contrainte reliant v1 à v2 (Figure 57).

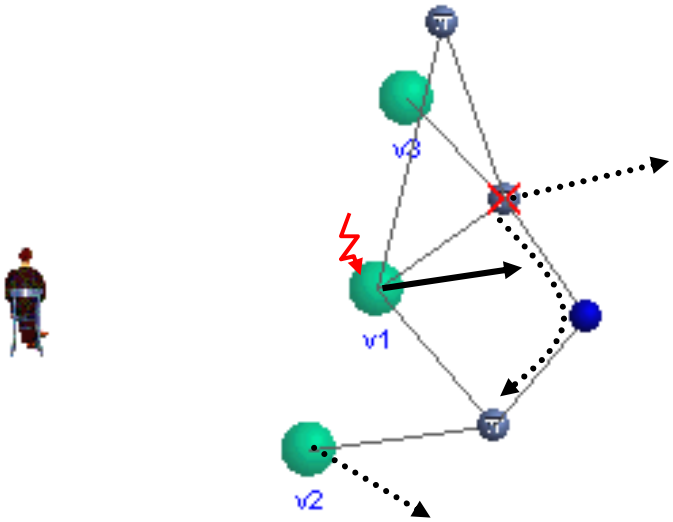


Figure 57 : exemple d'utilisation de la contrainte "le plus proche seulement" (2/3)

La contrainte reliant v1 à v3 est alors mutée, tandis que celle reliant v1 à v2 est dé-mutée. Si l'on continue à écarter v1 de l'avatar, v3 reste immobile et c'est v2 qui continue à suivre le mouvement de v1.

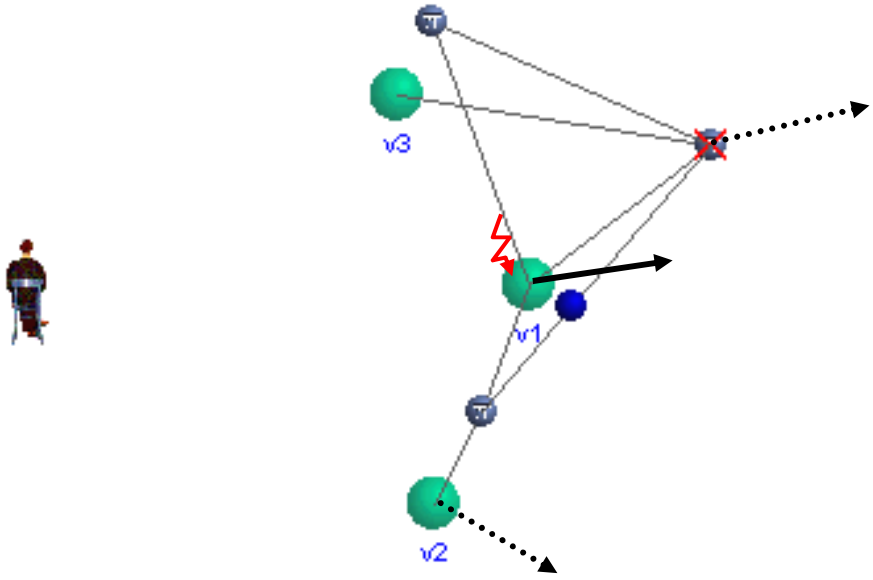


Figure 58 : exemple d'utilisation de la contrainte "le plus proche seulement" (3/3)

Pour résumer, nous décrivons par un graphique en Figure 59 les distances relatives de v2 et v3 par rapport à l'avatar lorsque v1 est déplacé. Dans ce schéma, d1 = distance(v1,avatar), d2 = distance(v2,avatar) et d3 = distance(v3,avatar). Notons que les coefficients directeurs des demi-droites ainsi que l'abscisse d_{limite} sont donnés par les conditions initiales du système, à savoir les positions des objets et contraintes au moment où l'algorithme est activé.

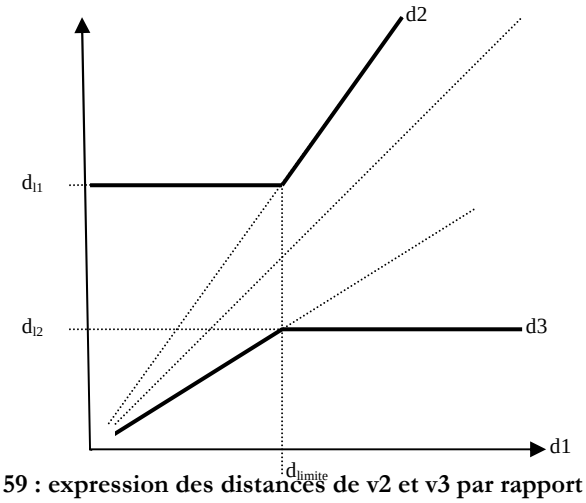


Figure 59 : expression des distances de v2 et v3 par rapport à l'avatar en fonction de la distance de v1 à l'avatar.

L'ensemble des contraintes mises en œuvre peut donc se résumer à une contrainte définie sur des intervalles de la manière suivante :

- Pour $d_1 \in]0, d_{\lim ite}]$: $\frac{d_1}{d_3} = Cste$
- Pour $d_1 \in [d_{\lim ite}, +\infty[$: $\frac{d_1}{d_2} = Cste$

5.1.4 Contraintes Animées

Plus marginale, la notion de contrainte « animée » décrit une contrainte qui génère elle-même des perturbations vers les objets contraints, suivant une cadence qui lui est propre, plutôt que d'attendre ces perturbations depuis un de ses objets contraints. L'idée sous-jacente à la réalisation des contraintes animées est qu'un « bon » mixage n'est pas forcément un mixage statique, et que dans certaines situations il est possible d'envisager qu'une partie des sources sonores évolue de manière autonome dans le temps, soit de façon déterministe, soit de façon libre (aléatoire) mais en gardant un contrôle sur l'étendue de ces variations.

5.1.4.1 Contraintes de rotation et déplacement rectiligne

Deux exemples de contraintes animées déterministes ont été intégrés au système, la contrainte de rotation et la contrainte de déplacement rectiligne. La contrainte de rotation (Figure 60) produit des perturbations périodiques vers ses objets contraints en leur imposant un mouvement de rotation dont elle est le centre avec un pas angulaire et une cadence que l'utilisateur peut détérminer.

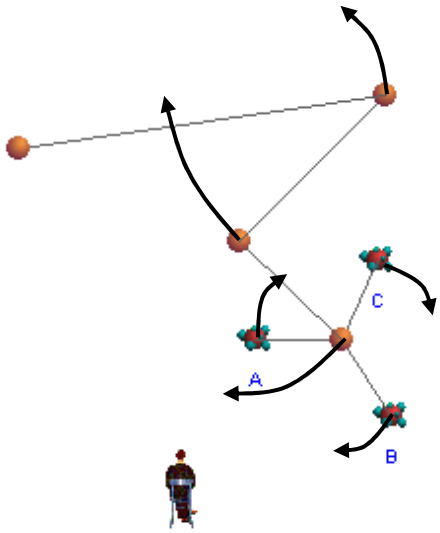


Figure 60 : exemple de combinaison de contraintes de rotation

De manière similaire, la contrainte de déplacement rectiligne (Figure 61) est battie à partir de deux objets graphiques quelconques (ici des drapeaux, objets servant uniquement à spécifier une position géométrique) qui déterminent les limites de la trajectoire et effectue des mouvements de va et vient entre ces deux repères au moyen d'interpolations dont le pas et la cadence sont choisis pas l'utilisateur. Dans notre exemple, l'objet V_1 est relié à la contrainte de déplacement rectiligne au moyen d'une contrainte de translation et effectue ainsi des mouvements périodique de gauche à droite de l'avatar.

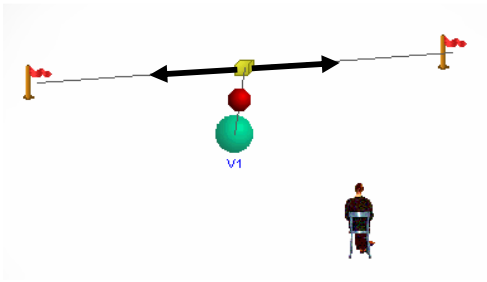


Figure 61 : exemple de contrainte de déplacement rectiligne

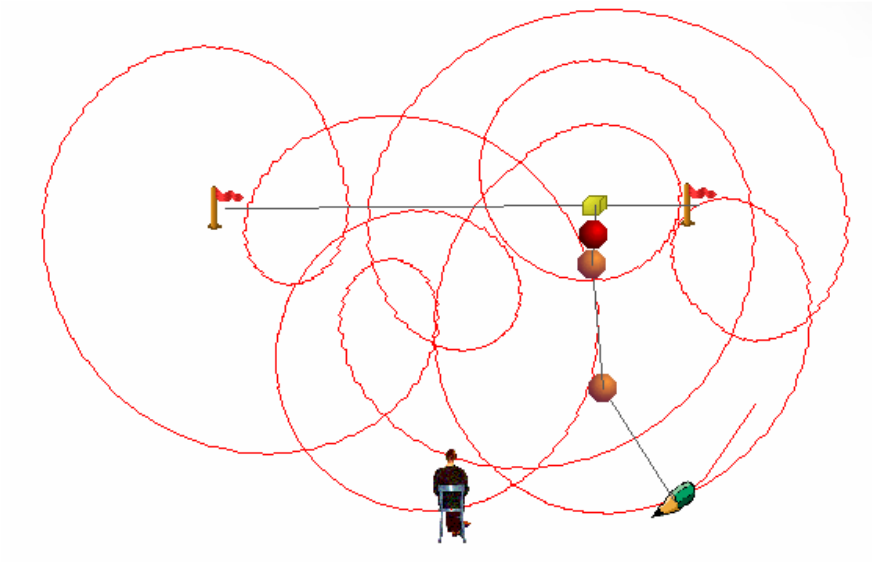


Figure 62 : exemple de trajectoire complexe construite à partir de la combinaison de contraintes animées

Pour finir, la Figure 62 donne un exemple de trajectoire relativement complexe, construite au moyen de la combinaison d'une contrainte de déplacement rectiligne, de deux contraintes de rotations et d'une contrainte de translation pour effectuer la jonction. L'objet contraint utilisé était spécialement ajouté au système pour laisser la trace graphique de ses déplacements.

5.1.4.2 Contrainte de Barycentre

La contrainte de Barycentre n'est pas une contrainte animée. Elle sert à représenter une variable intermédiaire au système afin de construire des relations plus sophistiquées portant non plus sur une les sources de manière individuelles mais sur les propriétés d'un ensemble de sources sonores. La Figure 63 présente un exemple de contrainte de barycentre. Affectée à trois sources différentes, elle se positionne systématiquement sur la moyenne des coordonnées de ses objets contraints. Lorsqu'une de ses variable est modifiée, la méthode de perturbation renvoie la valeur « vrai » si la contrainte a pu avec succès mettre à jour ses coordonnées et la valeur « faux » sinon.

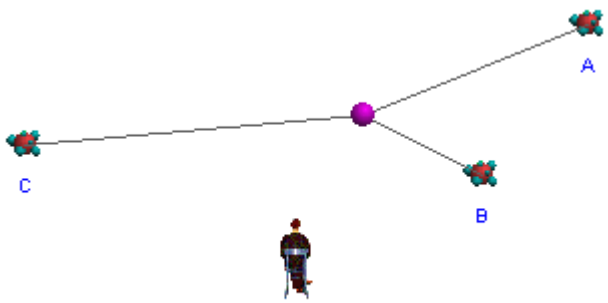


Figure 63 : contrainte de barycentre

L'utilisateur n'a pas la possibilité d'agir sur cette contrainte. Celle-ci devient donc intéressante lorsqu'elle est elle-même contrainte : soit par une contrainte de limite, par exemple, comme nous le présentons sous forme d'exemple dans le paragraphe suivant, soit par une contrainte géométrique, afin d'agir sur un paramètre à partir de la moyenne d'un ensemble de variables.

5.1.4.3 Contrainte de déplacement aléatoire

La contrainte de déplacement aléatoire choisit aléatoirement un point de destination à l'écran et tente de s'y rendre progressivement – à une cadence qui lui est propre et qui peut être ajustée par l'utilisateur – entraînant avec elle, par translation, l'ensemble de ses objets contraints.

Lorsque la contrainte parvient à son but ou lorsqu'elle rencontre une impossibilité (une contradiction imposée par une contrainte de limite par exemple, soit de manière directe soit comme résultat d'une propagation de perturbation au travers d'un de ses objets contraints), un nouvel objectif est choisi et le déplacement correspondant est amorcé.

La Figure 64 décrit un exemple d'utilisation de cette contrainte, combinée à une contrainte de barycentre et deux contraintes de limite. Cinq objets variables, nommés v1 à v5, sont connectés chacun à une contrainte de déplacement aléatoire. Ils tentent donc théoriquement tous de se déplacer indépendamment les uns des autres dans la zone de travail. Par ailleurs ils sont reliés entre eux par une contrainte de barycentre, qui elle est limitée en déplacement par deux limites radiales d'une part et deux limites angulaires d'autre part.

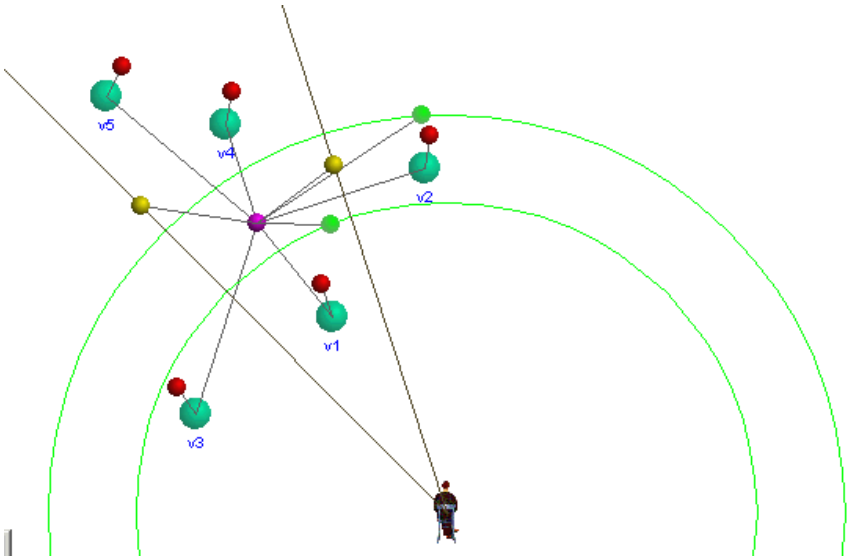


Figure 64 : combinaison de contraintes de déplacement aléatoire et d'une contrainte de barycentre

Lorsque les déplacements aléatoires sont activés, chaque objet v1,…, v5 se déplace librement choisissant une destination au hasard et tentant de s'y rendre en procédant par petit pas : les contraintes de déplacement aléatoire transmettent ces variations au objets sous la forme d'une perturbation comme si elle provenait de l'action d'un utilisateur. Les objets soumettent alors ces perturbations aux autres contraintes dont ils dépendent par le biais de l'algorithme de propagation : le résultat dans cet exemple est la mise à jour de la position de la contrainte de barycentre. La propagation continue et la contrainte de barycentre confronte sa nouvelle position avec les contraintes de limites auxquelles elle est attachée. Lorsque la contrainte de

barycentre dépasse une des limites, son déplacement est refusé par l'algorithme de propagation qui à tour refuse le déplacement de l'objet précédent ainsi que la perturbation initiale provenant de la contrainte de déplacement aléatoire. Celle-ci choisit alors un nouvel objectif de destination et tente à nouveau de s'y rendre.

Ce type de montage permet donc de mettre en scène un ensemble de sources sonores qui évoluent aléatoirement dans l'espace auditif, mais dont on peut garantir que la moyenne des distances par rapport à l'avatar reste comprise dans une plage de variations déterminée. L'exemple se prête donc particulièrement bien à la réalisation d'ambiances sonores où typiquement le mixage entre les différentes sources doit rester globalement constant mais où des petites modifications (qui préservent un niveau sonore globalement constant) peuvent apporter des variations intéressantes et enrichisantes au résultat sonore.

Pour finir, nous précisons d'une part que la contrainte de déplacement aléatoire ne met en œuvre qu'un processus aléatoire pariculier : beaucoup d'autres sont envisageables et peuvent proposer des propriétés intéressantes. D'autre part, la contrainte de barycentre n'est qu'un exemple de « variable intermédiaire » parmi beaucoup : son implantation ouvre la voie à d'autres qui pourraient représenter, pour un ensemble d'objets, le rayon minimum, le rayon maximum, la valeur X ou Y minimum, moyenne ou maximum,… chacune de ces variables pouvant être un moyen de contrôle intéressant sur la globalité d'un ensemble de sources sonores.

5.2 Algorithme de propagation de MusicSpace

Le problème de contraintes posé par MusicSpace est complexe : l'inventaire des contraintes effectué précédemment suggère l'utilisation simultanée de contraintes non-fonctionnelles, non-linéaires, à sorties multiples, à sens multiples, et surtout en présence de cycle dans le graphe.

Comme nous l'avons vu au Chapitre 3, il n'existe pas d'algorithme permettant de répondre entièrement et de manière complète à ce cas de figure. De plus le problème posé par ce cadre général est un problème difficile : il a été prouvé qu'il n'est pas possible de le résoudre en temps polynomial, ce qui pose des problèmes quand à la réactivité du système.

Enfin, l'algorithme existant qui s'approche au mieux du problème que l'on souhaite résoudre est également extrêmement complexe : il s'agit probablement d'Ultraviolet (voir section 3.4.10), dont le rôle est de diviser le graphe en sous sections de nature homogènes et de mettre en relation des solvers de contraintes spécifique pour traiter individuellement chacune de ces sous-section. L'utilisation de cet algorithme supposerait également de construire une implémentation des sous solvers qu'il utilise, Blue, Indigo, Purple et Deep Purple…

Nous avons donc décidé de construire un algorithme ad hoc, laissant de coté la complétude mais répondant de manière satisfaisante au problème que l'on se souhaite résoudre. L'algorithme repose en grande partie sur les concepts provenant d'algorithmes existants tels que Skyblue et Deltablue, mais se base également sur une heuristique propre au domaine que nous traitons et résulte en une certaine forme de continuité des contraintes : nous souhaitons en effet qu'une petite perturbation entraîne une petite propagation de sorte à éviter qu'une source sonore ne disparaisse d'un endroit de la scène sonore pour réapparaitre à un autre endroit comme conséquence d'une propagation. De tel sauts produiraient inévitablement des clics dans le résultat sonore.

5.2.1 Description de l'algorithme de contraintes

Tout comme dans les approches classiques de la propagation locale, le déroulement de l'algorithme de contrainte commence par une opération de perturbation, apportée au système par exemple par l'action de l'utilisateur sur l'un des objets. L'objet « perturbé » effectue une phase de propagation en notifiant toutes les contraintes auxquelles il est rattaché de la modification de sa valeur de position. A leur tour, les contraintes gèrent la propagation par un ensemble de perturbation vers les autres objets contraints de manière à rétablir la relation qu'elles représentent. L'algorithme de contraintes se déroulent donc en une succession de phases de propagation et de perturbation entre contraintes et objets, jusqu'à ce que soit toutes les perturbations sont acceptées (les objets et contraintes renvoient alternativement la valeur vrai comme valeur de retour) soit l'on rencontre une contradiction (une contrainte de limite est franchie, ou un objet reçoit plusieurs perturbations incompatibles).

Avant d'entrer dans le détail de l'algorithme de contraintes de MusicSpace, nous présentons la structure de la représentation interne, orientée objet, des données et surtout les interconnexions entre les différents objets au sein de l'arbre d'héritage des classes tel que représenté en Figure 65.

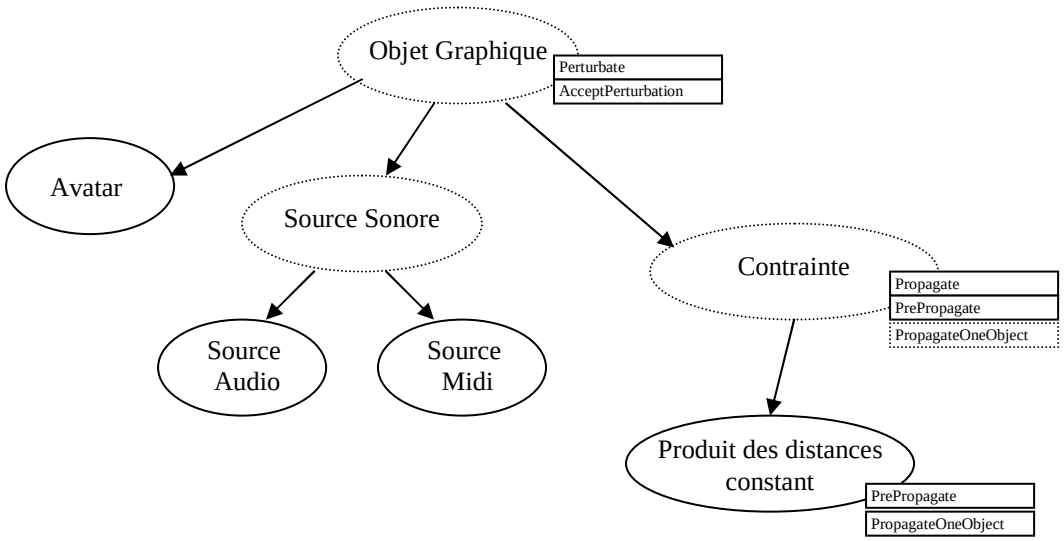


Figure 65 : extrait de l'arbre d'héritage des classes objets entrant dans la composition de scènes sonores dans MusicSpace

L'objet de base est nommé « objet graphique ». Celui-ci possède les propriétés et les méthodes nécessaires à sa visualisation dans l'interface graphique utilisateur, à savoir entre autres : une icône, des coordonnées géométriques et une méthode de représentation. Tout les objets qui en dérivent peuvent donc être représentés visuellement et si besoin redéfinir l'icône pour la remplacer par une autre qui leur est propre. De cet objet dérive l'ensemble des éléments de la scène tels que l'avatar, les sources sonore (par exemple audio et midi) et les contraintes. Dans cet hiérarchie, les classes Objet Graphique, Source Sonore et Contrainte sont déclarées abstraites (représentation de l'ovale en pointillés sur le schéma) : cela signifie qu'il n'est pas autorisé d'en construire des instances. Seules les classes terminales telles que « Source Audio » ou « Produit des distances constant » peuvent donner lieu a la création d'objets. En revanche ces classes définissent des méthodes que toutes les sous-classes pourront utiliser comme les

deux méthodes essentielles de notre algorithme, les méthodes « perturbate » et « propagate » que nous décrivons maintenant.

Les objets reçoivent les perturbations par le biais de la méthode « perturbate » dont les arguments sont le rapport de distance à appliquer, l'écart angulaire et l'origine de la perturbation. Le rapport de distance (appellé « ratioRayon ») intervient comme un facteur multiplicatif de la distance de l'objet perturbé par rapport à l'avatar. Par rapport si un objet reçoit une perturbation avec un ratioRayon égal à 2, celui-ci doit atteindre une nouvelle position qui double sa distance par rapport à l'avatar. La perturbation angulaire, elle s'exprime comme un écart angulaire (nommée deltaAngle) qui s'ajoute à la valeur d'azimut de l'objet par rapport à l'avatar.

Nous tentons de donner dans la suite de ce paragraphe une description simplifiée des méthodes essentielles de notre algorithme.

La méthode Perturbate commence par vérifier si l'objet a déjà été perturbé (par une autre contrainte) et si la perturbation qu'elle

```
methode perturbate(ratioRayon, deltaAngle, originePerturbation) {
  resultat <- VRAI ;

  // TESTER LES CYCLES :
  si ( getRatioRayon() != 1 && ( getRatioRayon() != ratioRayon ))
    // l'objet a déjà été déplacé en distance et la nouvelle perturbation
    // est incompatible avec l'ancienne
    retourner FAUX ;
  }
  si ( getDeltaAngle() != 0 && ( getDeltaAngle() != deltaAngle ))
    // l'objet a déjà été déplacé en azimut et la nouvelle perturbation
    // est incompatible avec l'ancienne
    retourner FAUX ;
  }


  // CALCULER LES NOUVELLES VALEURS DE POSITION
  (x,y)<- calculeNouvellesCoordonées(ratioRayon, deltaAngle) ;

  // EFFECTUER LA PHASE DE PROPAGATION
  pour toutes les contraintes cnt tel que :
    cnt.isAttachedConstraint(THIS)
    cnt != originePerturbation
  faire :
    resultat = resultat ET cnt.propagate(THIS)

  fin pour
  retourner resultat ;
}
```

Figure 66 : Algorithme de contrainte au niveau d'un objet.
Méthode « Perturbate »

Au niveau des contraintes, l'algorithme se situe essentiellement au sein des méthodes Propagate et PropagateOneObject. La méthode Propagate est la partie de l'algorithme commune à toutes les contraintes. Elle est définie dans la classe « contrainte ». Au contraire, la méthode PropagateOneObject correspond au minimum de code qu'il est nécessaire de différencier entre les différentes contraintes. Cette méthode est donc définie au sein de chacune des contraintes du système.

```
methode propagate(source ) {

  resultat <- VRAI ;
  prepropagate(source);

  // Appelle la méthode de propagation pour chacun des objets contraints
  pour tous les objets obj tel que :
    obj.estObjetContraint(THIS)
    obj != source
  faire :
    resultat <- resultat ET propagateOneObject(source, obj)
  finpour

  retourner resultat;
}


methode propagateOneObject(source, destination ) {
  // Cette methode décide, a partir des perturbations observées sur
  // l'élément source, les perturbations à renvoyer vers l'objet obj
  // (un des objets contraints) de sorte à rétablir la relation
  // spécifiée par la contrainte.

  // cette contrainte n'a aucun effet (toujours satisfaite). Voir les
  // exemples donnés plus bas

  retourner VRAI
}
```

Figure 67 : Algorithme de contraintes au niveau d'une contrainte. Méthodes « Propagate » et « PropagateOneObjet »

La méthode « PrePropagate » est une méthode propre à la contrainte, qui peut être appellée avant de commencer les propagations et donc de modifier les valeurs de position des différents objets contraints. Cette méthode est surtout utilisée dans la deuxième amélioration de l'algorithme que nous avons partiellement implémentée dans notre système, et présentons à la section 5.2.3.

Nous présentons maintenant une série d'exemples d'implémentation de la méthode propagateOneObject correspondant à différent types de contraintes.

Pour la contrainte « produit des distances constant », typiquement non fonctionnelle puisqu'il existe une multitude de manière de la satisfaire, nous proposons une implémentation fonctionnelle qui choisit une façon particulière de la résoudre : celle qui minimise les perturbations propagées aux objets contraints en réponse à une perturbation initiale. Ainsi, comme montré en Figure 46, si cette contrainte contrôle n objets, et mesure une perturbation α sur l'un de ses objets contraints, l'heuristique consiste à propager des perturbations homogènes aux n-1 objets restant, de valeur $\left(\frac{1}{\alpha}\right)^{\frac{1}{n-1}}$.

Nous remarquons dans cette implémentation que la contrainte n'ayant pas d'effet au niveau de la position en azimut des objets contraints, le deuxième argument de l'appel à la fonction perturbate pour l'objet destination est destination.getDeltaAngle(), c'est-à-dire que l'on perturbe au niveau angulaire l'objet destination par la valeur de perturbation angulaire qu'il connaît déjà. L'algorithme ignore alors simplement la perturbation en angle et ne décèle pas de conflit.

```
methode propagateOneObject(source,destination ) {

  // ELIMINER LE CAS OU LA SOURCE N'A PAS BOUGÉ EN DISTANCE
  // (RIEN A FAIRE)
  SI ( ! source.ratioHasChanged() ) retourner VRAI

  // ON CALCULE LE NOMBRE D'OBJETS QUI PARTAGENT LA PERTURBATION
  // ET LA PERTURBATION QUI EN RESULTE.
  nbObjectsToMove = THIS.nbConstrainedObjects – 1
  ratioRayon = ( 1 / source.getRatioRayon() ) ^ (1 / nbObjectsToMove)

  // APPLIQUER LA PERTURBATION
  resultat = destination.perturbate(ratioRayon,destination.getDeltaAngle(),THIS)
  retourner resultat

}
```

Figure 68 : Méthode « PropagateOneObject » pour la contrainte « produit des distances constant »

Dans le cas de la contrainte « rapport des distances deux à deux constants », la méthode propagateOneObject et excessivement simple, puisqu'elle revient à transmettre la valeur ratioRayon de l'objet perturbé à chacun des autres objets contraints.

```
methode propagateOneObject(source,destination ) {

  // ELIMINER LE CAS OU LA SOURCE N'A PAS BOUGÉ EN DISTANCE
  // (RIEN A FAIRE)
  SI ( ! source.ratioHasChanged() ) retourner VRAI

  // APPLIQUER LA PERTURBATION
  resultat = destination.perturbate(source.getRatioRayon(),
                                    destination.getDeltaAngle(),
                                    THIS)
  retourner resultat
}
```

Figure 69 : Méthode « PropagateOneObject »
pour la contrainte « rapport des distances deux à deux constants »

Les contraintes de limite ne font que vérifier l'inégalité. Leur méthode de propagation est donc relativement simple : elle se réduit à retourner la valeur VRAI ou FAUX suivant que le déplacement de l'objet pertubé viole ou non l'inégalité. Nous donnons à titre d'exemple la méthode de propagation pour la contrainte de limite radiale.

```
methode propagateOneObject(source,destination ) {

  // ELIMINER LE CAS OU LA SOURCE N'A PAS BOUGÉ EN DISTANCE
  // (RIEN A FAIRE)
  SI ( ! source.ratioHasChanged() ) retourner VRAI

  // VERRIFIER L'INEGALITÉ
  v1 <- THIS.getDistance() – source.getDistance()
  v2 <- THIS.getDistance() – ( source.getDistance() * source.getRatioRayon() )
  CROSSED <- ((V1 * V2 )< 0)
  SI ( CROSSED )
    // l'objet contraint est passé de l'autre coté de la limite.
    retourner FAUX
  SINON
    // L'objet contraint est toujours du même coté.
    retourner VRAI
  FINSI
}
```

Figure 70 : Méthode « PropagateOneObject »
pour la contrainte de « limite radiale »

Dans la réalité, la méthode de propagation de cette contrainte est plus complexe que celle présentée en Figure 70 étant donné les différents modes d'utilisation que peut prendre cette variable : comme nous l'avons mentionné au paragraphe 5.1.2, les contraintes de limites dans MusicSpace peuvent adopter différents comportements suivant que l'utilisateur décide si la limite peut être ou non repoussée par un objet, et si la contrainte peut ou non pousser un objet. Ces comportements se retrouvent directement dans la méthode de propagation et nécessite d'effectuer des tests sur la source de la perturbation et la destination de la propagation.

5.2.2 Première amélioration de l'algorithme

Nous avons montré comment établir des relations entre les sources sonores et l'avatar. Des exemples précis d'utilisation sont donnés dans la troisième partie de ce document, comme par exemple pour le morceau « TrioJazz » où un ensemble de contraintes portant sur trois sources permet de construire une scène sonore dans laquelle l'utilisateur peut modifier « sans risque » la position de chacun des instruments : en particulier, les contraintes permettent d'éviter les situation incohérentes mises en évidence au Chapitre 4.

Pourtant, la notion de « source sonore » n'est pas forcément, ni même toujours, l'entité élémentaire de contrôle la plus appropriée : déplacer un instrument ne fait pas toujours sens pour l'utilisateur, par exemple lorsque cet instrument fait partie d'une section instrumentale indivisible. De ces relations entre instruments émergent des notions plus haut niveau, telles que des « sections de cordes », « sections rythmiques », qui elles peuvent être intéressantes à manipuler. Ce sont ces notions là, combinaisons d'un ensemble de paramètres plus élémentaires que nous proposons de mettre à disposition de l'utilisateur au moyen d'objets particuliers nommés « handles » qui permettent de les matérialiser.

Pour ce faire, nous proposons une extension de l'algorithme de contrainte qui consiste à simuler le principe des contraintes unidirectionnelle à partir de notre algorithme de contraintes multidirectionnelles : ce type de simulation était déjà connu et avait été déjà démontré ([Sannella & Al., 1993b]). Dans le cadre de notre implémentation, il s'agit simplement de permettre à l'auteur de spécifier pour une contrainte donnée lesquelles de ses variables vont peuvent jouer le rôle de variables d'entrée et lesquelles peuvent servir de variables de sortie.

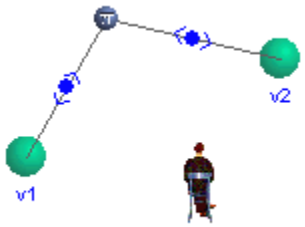


Figure 71 : Le mode par défaut des variables est "entrée - sortie"

Par défaut, toutes les variables des contraintes sont considérées comme des variables d'entrée et de sortie. Cet état peut être représenté graphiquement dans l'interface par des flèches situées entre la variable et la contrainte

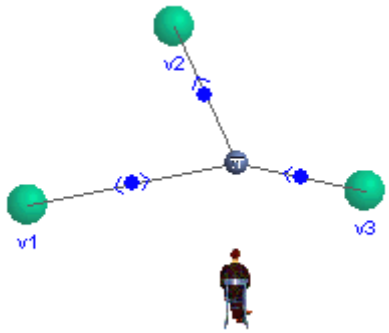


Figure 72 : Représentation graphique des trois états possibles d'une variable

Dans cet extension de l'algorithme de base, l'état de chacune des variables peut être permuté indépendamment des autres par un clic de souris et passe de manière cyclique par les trois états possibles :

- variable d'entrée uniquement, représenté par une flèche orientée de la variable vers la contrainte.
- variable de sortie uniquement, représenté par une flèche orientée de la contrainte, vers la variable
- variable d'entrée – sortie, représenté par une double flèche.

Dans l'exemple présenté sur la Figure 72, v1 est en mode « entrée sortie », v2 est en mode « sortie uniquement » et v3 en mode « entrée uniquement. Le tableau suivant exprime le résultat de la propagation, pour l'exemple de la Figure 72, c'est-à-dire une contrainte de type « produit des distances constant » et une perturbation de distance d'un facteur α appliqué respectivement à v1, v2, v3.

	Effet de la propagation

Variable perturbée	V1	V2	V3
V1		$\frac{1}{\alpha}$	X
V2	X		X
V3	$\left(\frac{1}{\alpha}\right)^{\frac{1}{2}}$	$\left(\frac{1}{\alpha}\right)^{\frac{1}{2}}$	

Dans le premier cas, lorsque v1 est perturbée, la seule variable de sortie disponible est v2 (v3 est marquée comme « entrée seulement » et v1 est l'origine de la perturbation. L'intégralité de la perturbation α est donc compensée par v2 par un facteur inverse.

Dans le deuxième cas, lorsque v2 est perturbée, aucune perturbation n'est propagée puisque v2 n'est vue que comme variable de sortie de l'algorithme.

Finalement, dans le troisième cas, nous retrouvons le comportement générique puisque en la perturbation apportée à v3 est répartie de manière égale parmi les deux autres variables marquée comme variables de sortie.

Dans le cas où la contrainte de disposerai d'aucune variable de sortie la propagation est simplement acceptée. Ce mécanisme est un parti pris qui permet d'une part une division par 0 en tentant de répartir la perturbation sur les variables de sortie, mais aussi de tirer plus amplement partie du concept de « handle » que nous présentons maitenant.

Ce mécanisme de contraintes unidirectionnelle permet d'introduire la notion de « handle » équivalente à une variable d'entrée du système : cette variable joue le rôle de paramètre dont la modification aura des répercussions sur un ensembe de sources sonores par exemple. Ces handle sont représentés graphiquement comme les autres composants de la scène, soit par leur icône par défaut dérivant un curseur, soit par une icône choisie par l'auteur.

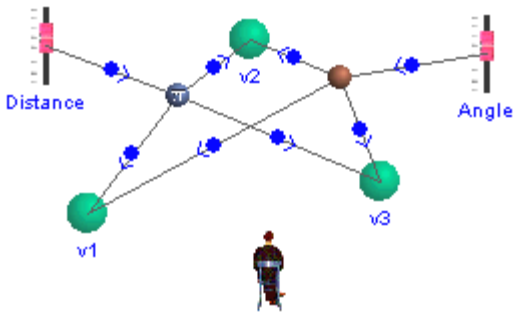


Figure 73 : exemple d'utilisation du mode "one-way"

L'exemple présenté en Figure 73 montre une utilisation possible du mode one-way associé à la notion de handle : les trois variables V1, V2, V3 peuvent être contrôlées au moyen de deux handles nommés « distance » et « angle » par l'intermédiaire de deux contraintes « rapport des distances deux à deux constants » pour la première et « différences angulaires constantes » pour la deuxième. Ainsi, chaque variable n'étant que variable de sortie de contraintes peut être déplacée librement comme pour « ajuster » la scène. Une fois l'équilibre atteint – les variables sont par exemple à des distances relatives statisfaisantes par rapport à l'avatar – il est possible de continuer d'agir sur la scène en gardant les proportions de distance constantes au moyen du handle. Il en est de même pour les écarts angulaires et le handle « angles ».

Pour un ingénieur du son, cela permettrait par exemple d'ajuster indépendamement la position de sources sonores au sein d'une section instrumentale et ensuite d'agir de manière globale sur cette section instrumentale au moyen d'un handle et de contraintes unidirectionnelles. Pour un utilisateur non expérimenté les différentes sources sonores de la section instrumentale seraient alors masquées pour ne laisser le contrôle que sur le handle lui-même. Notons que le handle répond de manière identique à une source sonore dans notre système et en particulier qu'il peut à son tour être contraint par des contraintes de limite par exemple, afin de limiter sa plage admissible de variations.

Les contraintes unidirectionnelles ainsi que la notion de handles sont mises en pratique dans la troisième partie de ce document, relative à l'éxpérimentation. En particulier la configuration « dynamique » de Ken Mouka, présenté au Chapitre 8 donne des exemples d'utilisation « musicales » des handles. Le travail de Nicolas Deflache, rapporté au Chapitre 10 fait lui aussi état de l'utilisation de ces techniques ([Deflache, 2001]).

5.2.3 Deuxième amélioration possible de l'algorithme

L'algorithme est incomplet : il n'aboutit pas systématiquement, dans les cas difficiles, à une solution même lorsqu'il en existe. Mais il est un ensemble de cas de complexité intermédiaire, mettant en œuvre des contraintes n-aires et où l'algorithme refuse la perturbation alors qu'il serait possible de calculer facilement une réponse satisfaisante. Ceci est du à deux points clefs propres à notre algorithme : tout d'abord les contraintes de limite sont simplement vérifiées et n'ont que deux valeurs de retour possibles « vrai » ou « faux » suivant que la perturbation a été acceptée ou refusée. D'autre part l'implémentation « fonctionnelle » des contraintes non fonctionnelles telle que « produit des distances constant » pour laquelle nous tentons de répartir la perturbation initiale vers l'ensemble des objets contraints semble relativement limitative. Nous tentons d'établir dans cette section des pistes de recherche qui permettraient d'améliorer les résultats de l'algorithme sans pour autant entrer dans la complexité combinatoire du problème. Ces pistes n'ont pu être, pour l'instant, que partiellement implémentée dans notre système.

Le diagrame représenté en Figure 74 donne un tel exemple de contraintes qui échoue à trouver une solution pourtant assez élémentaire : trois variables v_1, v_2 et v_3 sont liées par une contrainte « produit des distances constant ». Par ailleurs v_3 est contrainte par une limite de distance à rester inférieure à une valeur constante.

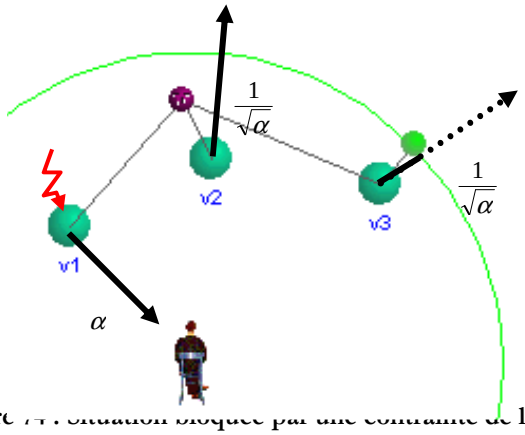


Figure 74 : situation bloquée par une contrainte de limite

En perturbant V_1 d'un par un facteur multiplicatif α, notre algorithme a pour consigne de compenser cette perturbation de manière équitable sur les variables V_2 et V_3, d'un facteur $\left(\frac{1}{\alpha}\right)^{\frac{1}{2}}$ de telle sorte que le produit des perturbations $\alpha \times \left(\frac{1}{\alpha}\right)^{\frac{1}{2}} \times \left(\frac{1}{\alpha}\right)^{\frac{1}{2}} = 1$. Si l'on rapproche V_1 de l'avatar, l'application de l'algorithme aura pour effet d'écarter v2 et v3 à concurrence de la limite imposée par la deuxième contrainte. A ce niveau, l'algorithme ne trouve plus de solution, dans le sens où il est impossible de continuer à compenser de manière égale la perturbation appliquée à V_1 sur les variables V_2 et V_3 tandis que le problème plus général énoncé par la contrainte (« produit des distances constant ») aurait encore des solutions. En particulier, si $\alpha_{3\max}$ est la perturbation maximum qu'admet v3, une solution consiste à transmettre la perturbation $\frac{1}{\alpha \times \alpha_{3\max}}$ (voirFigure 75).

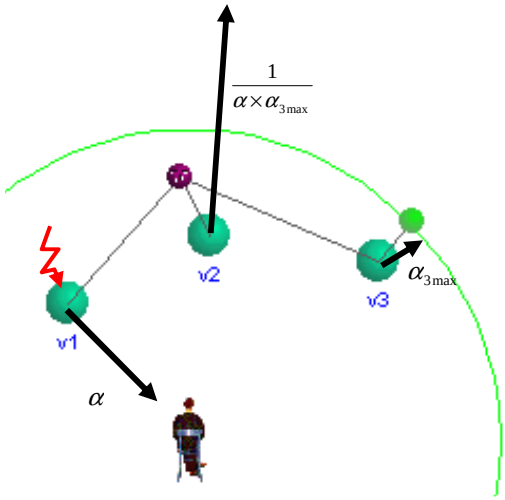


Figure 75 : solution proposée

L'amélioration de l'algorithme consisterait donc d'une part à traiter les contriantes d'inégalité de manière plus souples en proposant trois valeurs de retour « perturbation acceptée »,

« perturbation refusée » et « perturbation au mieux » dans laquelle la variable prend la valeur limite acceptable par la contrainte et un status spécial, spécifiant que la contrainte originale n'a pu être entièrement satisfaite.

D'autre part, pour une contrainte n-aire non-fonctionnelle, l'heuristique serait de collecter la perturbation globale, constituée de la perturbation initiale et des « valeurs au mieux » des variables partiellement satisfaites et de la propager de manière équitable au sein des variables toujours entièrement libres. Un tel algorithme ne serait toujours pas complet, mais resterai de complexité polynomiale et permettrait de débloquer un nombre important de situations.

Pour finir, une ultime amélioration consisterait à mémoriser au préalable les valeurs des rapports des distances de sorte que la déformation entraînée par une contrainte de limite puisse être réversible : bien que la contrainte de limite entraine une déformation des rapport de distance entre les objets, les rapports originaux pourraient être retrouvé en ramenant l'objet intialement perturbé vers sa position originale.

Chapitre 6
Inventaire des systèmes de spatialisation utilisés

Nous avons testé notre système dans différentes situations d'utilisation et en particulier en connexion avec différents moteurs de spatialisation, allant des plus simples, aux plus perfectionnés. Nous décrivons dans cette section l'ensemble de ces situations en tentant de mettre en évidence ce qui les distingue chacune, les unes des autres.

6.1 Spatialisation Midi :

Le premier cadre de spatialisation envisagé a été le contrôle de la position des signaux sonores produits par un synthétiseur. En particulier, nous nous sommes intéressé au cas de fichiers MIDI répondant au standard Général MIDI, c'est-à-dire pour lesquels il est possible de prédire, en fonction des indications de changement de programme, quel instrument acoustique le synthétiseur devra imiter.

Dans la norme Midi initiale, les fabricants de synthétiseurs avaient libre choix sur l'organisation des algorithmes de synthèse en regard de la table de changement de programme. Rien ne permettait donc d'affirmer qu'un fichier MIDI faisant appels à des numéros de changement de programme arbitraires serait restitué correctement sur un synthétiseur différent. Le standard General Midi ([General Midi, 1991]) tente de remédier à ce problème en définissant une table de 128 changements de programmes pour lesquels les instruments à « imiter » sont prédéterminés. Par exemple, le numéro 1 est un piano, le numéro 22 un accordéon, le 59 un tuba,…). Une table de correspondance a également été définie pour le canal de percussions, définissant des numéros de notes correspondant à des instruments de percussion prédéterminés (38 est une caisse claire, et 80 un son de triangle étouffé).

Cette information de numéro de programme présente dans les fichiers MIDI nous est précieuse : elle nous permet tout d'abord de représenter graphiquement les différentes sources à l'aide d'icônes correspondantes. L'utilisateur peut, par la représentation symbolique de l'instrument agir plus rapidement et plus sûrement sur la source sonore qu'il souhaite déplacer. Grâce à ces messages de changement de programme, un même canal MIDI peut donc servir à restituer plusieurs instruments à condition qu'ils ne jouent pas simultanément mais consécutivement. La Figure 76 représente un exemple de fichier MIDI complexe, faisant intervenir une vingtaine de sources. Les instruments inactifs apparaissent en ombre chinoise. Quand vient leur tour de jouer, ils échangent leur rôle avec l'instrument actif qui occupait précédemment le canal MIDI.

Figure 76 : exemple de fichier Midi complexe.
Plusieurs instruments partagent le même numéro de canal MIDI

La spatialisation MIDI n'est pas de bonne qualité : les seuls paramètres sur lesquels il nous est possible d'agir, sont les contrôleurs midi 7 et 10, agissant respectivement sur l'intensité et la valeur de panoramique de chaque canal, simple dosage de niveau entre les canaux gauche et droit des sorties stéréo des synthétiseurs. La perception de localisation n'est donc que très faiblement rendue et l'exercice relève donc davantage du mixage que de la spatialisation. Par ailleurs, la limitation fondamentale des fichiers MIDI est qu'ils interdisent complètement certains instruments acoustiques tels que la voix chantée, par exemple, ce qui interdit une part considérable du répertoire musical.

Pourtant, les fichiers MIDI présentent une information structurée, symbolique (économique), qui se prête donc bien à certaines manipulations intéressantes. La Figure 77 en donne un exemple : un choix judicieux de filtres symboliques nous a permis de scinder la partie de piano du trio de jazz en deux parties dites « main gauche » et « main droite ». Deux nouveaux objets « piano » sont représentés et affectés à des canaux MIDI différents, de telle sorte qu'il nous est possible de les spatialiser indépendamment l'un de l'autre.

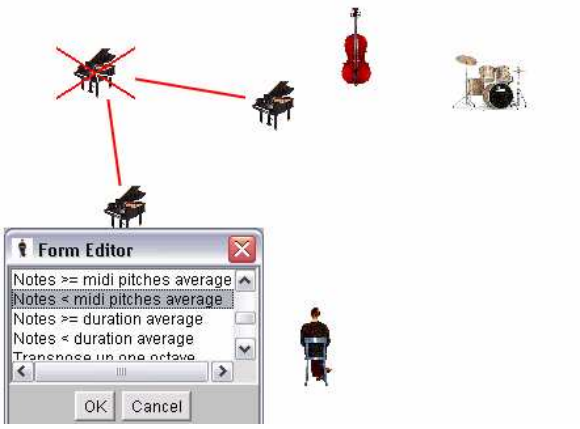


Figure 77 : division de la piste de piano du trio de Jazz,
en parties de main gauche et main droite

Il devient donc envisageable, dans cet exemple, de relier la partie de main gauche à la contrebasse et à la batterie puisque typiquement la main gauche du piano est une partie

d'accompagnement qui vient en opposition de la main droite, dite « soliste ». L'exemple du trio de jazz sera plus amplement détaillé dans la suite de ce document, au Chapitre 7, dans la partie expérimentation.

6.2 DirectX :

Le système DirectX de Microsoft a déjà été présenté dans la section 1.4.2 en particulier pour sa capacité à représenter un système d'interfaçage entre applications multimédia et spatialisateurs. C'est sa capacité à spatialiser les sons qui nous intéresse ici, et nous en proposons une mise en œuvre, au sein de MusicSpace, réalisée au cours d'une collaboration avec Peter Hanappe [Pachet & al, 2000] et [Pachet & al, 2000b]. Dans cette version de notre prototype, des objets graphiques spécifiques représentent des sources sonores de DirectSound3D. Ces objets sont attachés à un fichier audio et permettent, au travers de leur éditeur, de modifier l'ensemble des paramètres (en dehors des relations géométriques respectives) que DirectX permet d'ajuster. En dehors de la possibilité fondamentale d'utiliser des enregistrements vocaux (voir l'exemple présenté au Chapitre 8) la version audio de MusicSpace autorise également le mixage en formats autres que la stéréophonie, lorsque les cartes sonores utilisées le permettent, tels que la quadriphonie ou la diffusion de type Dolby Surround, enrichissant sensiblement l'expérience de spatialisation.

Par ailleurs, pour parvenir à lire un nombre important de pistes audio simultanément, nous avons choisi d'entrelacer celles-ci, réduisant ainsi les sauts de la tête de lecture du disque dur lors de la reproduction sonore. La technique consiste à mettre alternativement quelques échantillons de chacune des pistes dans un seul et même fichier plutôt que d'utiliser les pistes de manière individuelle (cf Figure 78 où trois pistes sont découpées en fragments pour être entrelacées). Ce format contraint relativement la lecture puisque chaque piste doit être obligatoirement lue et qu'il n'est pas possible d'en retarder une par rapport aux autres, mais rend possible l'exploitation de ces données sur CD-Rom par exemple, dont la tête de lecture est d'une lenteur et d'une mobilité telle qu'il serait inenvisageable de sauter alternativement entre plusieurs endroits du disque pour en extraire les informations de chacune des piste en temps réel.

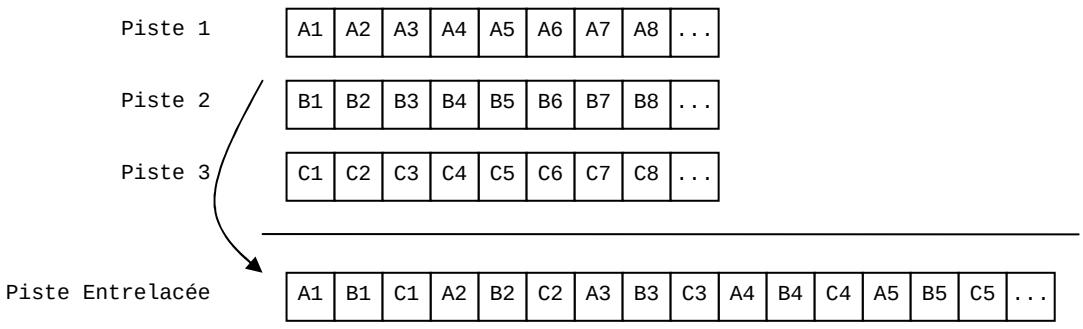


Figure 78 : exemple d'entrelacement de 3 pistes

Un intérêt majeur du système DirectX est qu'il est possible – à l'aide d'un ordinateur suffisamment performant – de jouer simultanément un nombre important de fichiers sonores

(des essais sur un ordinateur de bureau de type PC muni d'un processeur Pentium III cadencé à 450 Mhz ont montré qu'il était possible de jouer simultanément une trentaine de sources sonores sans constater de faiblesse du système). Cet avantage indéniable permet de mettre en œuvre des applications de la spatialisation intéressantes et inenvisageables dans les autres systèmes de part leur limitations soit matérielle dans le cadre du contrôle des consoles de mixage (voir section 6.3) de par leur nombre limité de pistes d'entrée, soit en coût de calcul dans le cadre par exemple du contrôle du spatialisateur IRCAM (voir section 6.4) pour lequel typiquement un ordinateur macintosh n'est capable de gérer qu'environ trois ou quatre sources lorsque l'on souhaite par exemple un rendu binaural avec la précision la plus grande.

6.3 Contrôle des consoles de mixages O2R / O3D :

Le contrôle de consoles de mixage a également été l'objet d'expérimentations dans notre étude. Notre choix s'est orienté initialement vers la console Yamaha O2R entièrement numérique, disponible au laboratoire SONY CSL. La console Yamaha O3D a également été étudiée lors du travail de recherche de Nicolas Deflache (voir Chapitre 10) au CNSM de Paris. Nous mettons ici en avant les différentes conditions d'utilisation de ces consoles qui ont mené à différentes implémentations au sein de notre système en essayant d'en dégager les caractéristiques essentielles.

6.3.1 Mixage Stéréophonique

La première situation de contrôle concerne la console Yamaha O2R (voir Figure 79) avec pour objectif le mixage au format stéréophonique, c'est-à-dire le format de restitution sonore auquel cette console est initialement conçue. Des objets spécifiques ont donc été introduits à notre système, correspondant à chacune des pistes d'entrée de la console. Ces objets ont la particularité non seulement de transmettre des valeurs de contrôle à la console relatifs au niveau et au panoramique des piste en fonction de leurs positions par rapport à l'avatar dans MusicSpace, mais aussi de recevoir des informations de contrôle en provenance de la console et de les manifester par des modifications de leur position. Plus précisément, l'angle sous lequel l'avatar voit les objets O2R sont traduits en valeur de panoramique pour la tranche correspondante et la distance de l'avatar aux objets O2R est relié au réglage de niveau de chaque tranche de telle sort qu'une distance très faible corresponde à un gain de 0db, et qu'une distance supérieure ou égale à la demi diagonale de l'écran équivaille à un gain de -∞.

Figure 79 : connexion de notre prototype MusicSpace et d'une console de mixage numérique Yamaha O2R

Il existe donc une complète correspondance entre les positions des objets graphiques dans MusicSpace et les valeurs des paramètres de la console que l'ingénieur du son utilise le plus fréquemment au cours du mixage. Cette mise en œuvre est relativement intéressante puisque il devient donc possible, au cours du mixage, de se passer complètement de MusicSpace et n'intervenir qu'au niveau de la console elle-même. Notre système sert alors en quelque sorte d'outil auteur permettant au préalable de définir les relations souhaitées entre les pistes de la console : en particulier, il est possible à l'aide du mécanisme de contraintes de définir entre des pistes le type de relations déjà connues du monde du mixage. Par exemple, l'utilisation d'une contrainte de type « rapport des distances deux à deux constant » entre différentes pistes d'entrée permet tout d'abord de lier celles-ci comme dans le cas d'un sous-groupe pour les paramètres de niveau. La même opération mais cette fois ci à l'aide d'une contrainte angulaire constante permet de construire le même résultat mais pour les paramètres de panoramique. Ces notions de base, déjà présentes dans les consoles peuvent être largement développées au sein de notre système en variant à volonté les contraintes utilisées et la façon de les organiser. Il est par exemple possible de reconstruire la notion de sous-groupes sur les sous-groupes eux même et finalement se donner une représentation arborescente des paramètres du mixage, en se donnant la possibilité d'intervenir à chaque nœud de l'arbre.

Par ailleurs les contraintes de limite apportent elles aussi à la console des fonctionnalités supplémentaires en limitant par exemple la course de manipulation des curseurs : lorsque l'utilisateur dépasse une certaine valeur d'amplification ou atténuation du signal, l'algorithme de propagation refuse la perturbation et renvoie à la console un message repositionnant le curseur à sa valeur précédente. En pratique, le curseur peut être manipulé librement dans la course admissible de valeurs, et « bloque » dès que l'on sort de celle-ci. Pour les paramètres de panoramique, le fonctionnement est différents : les contrôleurs présents sur la console sont des boutons rotatifs « sans fin » et ne sont pas motorisés comme les curseurs de niveau. Lorsque l'utilisateur tourne ce bouton, la valeur de panoramique est modifiée en conséquence jusqu'à ce qu'elle atteigne sa valeur minimum ou maximum. Dès lors, le bouton peut toujours être manipulé mais il n'a plus d'effet sur le paramètre qu'il contrôlait. Avec les contraintes angulaires dans MusicSpace le fonctionnement général reste le même, mais il devient possible de modifier les valeurs minimale et maximale acceptable pour le panoramique. En manipulant le bouton sur la console, l'utilisateur déplace un objet de manière radiale dans MusicSpace, par

rapport à la position de l'avatar : ce déplacement induit un changement de valeur correspondant pour le paramètre de panoramique de la console. Lorsque l'objet graphique, dans MusicSpace, rencontre une contrainte angulaire, le déplacement radial est refusé et la valeur de panoramique reste figée à la valeur limite, imposée par la contrainte.

Pour finir, notre système met également en évidence la notion de composition d'une scène sonore et de point de vue représenté par l'avatar. Lorsque l'utilisateur déplace la position de l'avatar, les paramètres de chacune des pistes sont mis à jour de manière cohérente et corrélée ce qui entretient et renforce cette même notion de scène. Dans un tel cas, même avec un nombre de sources très limité, le nombre de paramètres à ajuster simultanément devient alors beaucoup trop important pour qu'un ingénieur du son puisse en imaginer le pilotage manuellement.

6.3.2 Mixage Quadriphonique

Une deuxième série d'objets, les objets « O2R Quadra » rendent possible le mixage à partir d'une console O2R pour un format quadriphonique, c'est-à-dire un format constitué de quatre pistes, destinées à des haut-parleurs positionnés en carré autour de l'auditeur. Bien que la console Yamaha O2R n'ait pas été conçue dans cet objectif il est techniquement possible d'effectuer un tel mixage au moyen des bus dits « départs auxiliaires ». Ceux-ci sont des pistes de sortie supplémentaires traditionnellement utilisées pour alimenter les processeurs d'effet tels que les unités de réverbérations : chaque piste d'entrée de la console possède un réglage individuel et indépendant permettant de doser la quantité de signal qui doit être envoyée au processeur d'effet. Nous utilisons donc, dans cette mise en œuvre, les canaux auxiliaires 1 à 4 comme des pistes de sorties indépendantes alimentant les voies d'amplification respectivement des haut-parleurs avant-gauche, avant-droit, arrière-droit et arrière gauche. Le procédé de spatialisation reproduit est très simple du fait du peu de contrôles disponibles pour les canaux auxiliaires et revient à une situation de « panning d'amplitude » sur quatre haut-parleurs telle que décrite dans [Chowning, 1971] dans une version simplifiée puisque cet article préconise pour la simulation de la sensation de distance l'utilisation de modules de réverbération indépendants pour chacun des canaux de sortie afin de générer une réverbération diffuse au niveau spatial ce qui n'était dans notre cas matériellement pas possible. Cette méthode de spatialisation, bien que très simple, a tout de même l'avantage d'être particulièrement résistante dans une situation où l'on ne connaît pas la position et l'orientation exacte de l'auditeur, par rapport à d'autres méthodes faisant usage de retards et de modification de phases.

Ainsi, les objets « O2R Quadra » dosent pour chacune des pistes d'entrée les réglages de ses départs vers les pistes auxiliaires 1 à 4 en fonction de la distance à l'avatar, et de l'angle sous lequel celui-ci les voit. En pratique, la représentation cartésienne est plus adaptée et permet de calculer un pourcentage d'interpolation gauche-droite en projetant les coordonnées des sources sur l'axe horizontal passant par l'avatar et un pourcentage d'interpolation avant-arriere en projetant sur l'axe vertical passant par l'avatar. Les valeurs obtenues sont ensuite normalisées de sorte qu'un mouvement circulaire autour de l'avatar soit reproduit à niveau sonore constant.

La nécessité d'un système tel que le notre pour réaliser ce type de mixage est encore plus évidente que dans la situation de mixage stéréophonique décrite dans la section précédente : le déplacement d'une source sonore demande, à chaque nouvelle position, d'ajuster simultanément 4 paramètres d'amplitude. Lorsque c'est l'avatar qui est déplacé, ce sont les

quadruplets de paramètres de chacune des sources qui doivent être contrôlés simultanément.

6.3.3 Mixage au format « 5.1 »

Pour finir, le contrôle de la console Yamaha O3D, dans le cadre du travail de recherche de Nicolas Deflache (voir Chapitre 10) a permis d'étudier le cas du mixage en format 5.1, c'est-à-dire pour un dispositif de restitution muni de 5 haut-parleurs (un haut-parleur central, deux avants gauche et droits et deux arrières) et d'un caisson de restitution des fréquences basses. La console O3D est conçue initialement pour gérer différents formats de sorties multipistes comme en particulier le format 5.1. Inutile donc, comme dans le cas précédent, d'user de moyens artificiels pour utiliser la console dans un mode de fonctionnement auquel elle n'était pas destinée.

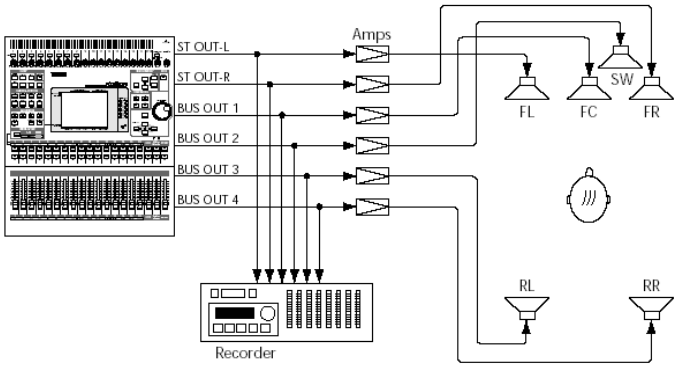


Figure 80 : connexions de la console O3D en mixage 5.1

Si matériellement la console a donc été adaptée aux nouveaux formats de diffusion actuels, l'interface de contrôle, le mécanisme utilisé, tel que le montre la Figure 80, est assez proche de celui que nous avions mis en œuvre lors du mixage en quadriphonie à l'aide de la console O2R : les pistes de sortie stéréo sont conservées pour les haut-parleurs avant gauche et droit, et quatre pistes de sorties auxiliaires sont utilisées pour alimenter les canaux correspondants aux autres haut-parleurs : central, arrières, et sub-woofer.

L'interface de contrôle correspondante proposée, représentée en Figure 81 permet de positionner individuellement chacune des sources dans la zone d'écoute, soit manuellement, soit au moyen d'un ensemble de primitives de base générant des trajectoires : il s'agit de mouvement simples avant / arrière, gauche / droite, en diagonale ou bien des mouvements rotatifs autour du point d'écoute.

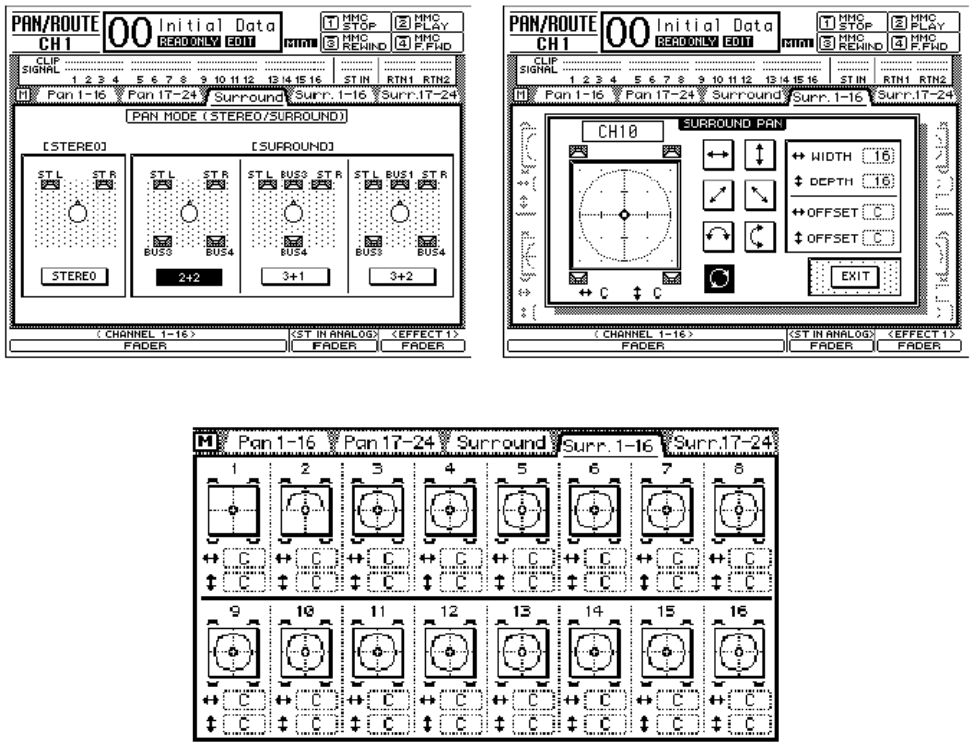


Figure 81 : fenêtres de contrôle des paramètres de « panning » de la console O3D

Pourtant, cette interface ne permet visiblement pas de tirer partie aux maximum des possibilités offertes par la console : en particulier, la présentation des paramètres de panoramique sous forme de « banc » à l'image des curseurs de réglage de niveau ne semble pas forcément la plus adaptée et serait peut être avantageusement remplacée par une représentation de la scène unique permettant d'évaluer graphiquement les relations spatiales entre chacune des pistes d'entrée. Par ailleurs, une seule source n'est accessible à la fois, et il n'existe pas la possibilité d'établir, comme nous le proposons dans MusicSpace au moyen des contraintes, de relations entre les sources sonores.

Nous n'étendons pas davantage la discussion dans cette section : le travail réalisé par Nicolas Deflache sera décrit amplement au Chapitre 10, dans la partie « Expérimentation » de ce document.

6.4 Pilotage du Spatialisateur IRCAM

Comme mentionné dans la section 2.2, le spatialisateur IRCAM (le Spat) a déjà fait l'objet d'études approfondies au niveau de son interface de contrôle : les paramètres élémentaires et bas niveau de la chaîne de traitement du signal ont été remplacés par un jeu plus restreint de paramètres perceptifs dont la signification est plus évidente à l'utilisateur. L'expérience de contrôle du spatialisateur IRCAM par le logiciel MusicSpace concerne la partie des paramètres du Spat décrivant les relations géométriques existantes entre chaque source sonore et l'auditeur.

Du point de vue de MusicSpace, l'intérêt d'une telle expérience relève de la qualité du traitement sonore et de la précision dans la perception de localisation rendue par le Spat. Par

ailleurs, par rapport aux objets MIDI initiaux, le Spat apporte également un ensemble de paramètres de contrôle plus riche, rendant l'exercice plus intéressant. En particulier, en dehors des paramètres d'azimut et de distance traditionnels déterminant la façon dont l'auditeur voit la source, le Spat tient compte également des paramètres opposés, décrivant la manière dont la source est orientée par rapport à l'auditeur, ainsi que la manière (simplifiée) dont cette source rayonne dans l'environnement.

Du point de vue du Spat, MusicSpace apporte des modalités de contrôle intéressantes. Au même titre que les interfaces « Circ » (Max/MSP) et « Circle » (jMax/FTS) (voir Figure 14 et Figure 15 page 35), MusicSpace permet tout d'abord de représenter dans un même environnement de contrôle plusieurs sources sonores à traiter, tandis que le SpatOper ne permettait de n'envisager le contrôle que d'une source à la fois, indépendamment des autres. Cette particularité permet d'introduire également la notion de « point d'écoute » commun à plusieurs sources, a priori inexistante dans le Spat. Les déplacements de l'avatar dans MusicSpace se répercutent localement comme des modifications relatives des distances et azimuts des sources au sein de chacun des modules Spats.

Par ailleurs des objets graphiques particuliers ont été ajoutés, de manière à représenter et contrôler graphiquement la manière dont les sources sont orientées par rapport à l'auditeur, et la façon dont elles rayonnent. Ces objets complètent les objets de type « source sonore du Spat » et permettent de basculer les chaînes de traitement correspondantes du mode « omnidirectionnel » vers le mode « directionnel » : la distance séparant la source de son objet directivité détermine si la source est omnidirectionnelle (distance faible) ou au contraire très directionnelle (distance élevée).

Figure 82 : deux sources sonores du Spat,
omnidirectionnelle (gauche) et directionnelle (droite)

Les principales relations géométriques entre un point d'écoute et une source sonore sont représentées en Figure 83 : l'azimut d'une source correspond à l'angle sous lequel l'avatar voit la source. Dans le cas du Spat, cette valeur s'exprime comme l'écart angulaire que fait la source avec l'axe de l'avatar. Au contraire, l'orientation d'une source, paramètre « Yaw » du Spat est la valeur angulaire en degrés décrivant sous quel angle la source voit l'auditeur dans le plan horizontal et s'exprime donc comme l'écart de l'avatar par rapport à la direction d'émission principale de la source.

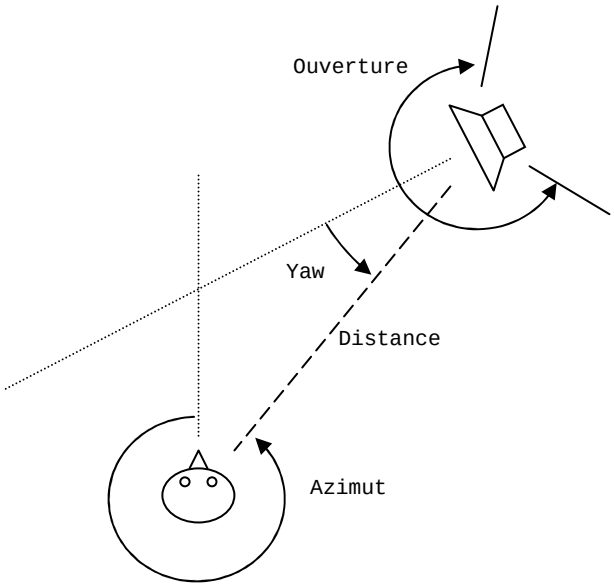


Figure 83 : représentation des relations géométriques (Distance, Azimut, Ouverture et Yaw) entre une source sonore et un auditeur

Le déplacement d'une source sonore implique donc le contrôle simultané d'un ensemble important de paramètres… à moins de spécifier, par exemple, des relations entre la position de la source sonore et ses autres paramètres géométriques tels que son orientation. Nous voyons ici un exemple naturel d'application direct ne notre système de contrainte : la Figure 84 montre comment différentes relations peuvent être établies entre les sources sonores et l'objet de directivité qui lui correspond. Dans le premier cas, à gauche sur la Figure 84, une contrainte de translation est utilisée entre les deux objets : l'ensemble de ces objets se déplace donc toujours solidairement, la source est toujours orientée dans la même direction et son paramètre de directivité reste constant également.

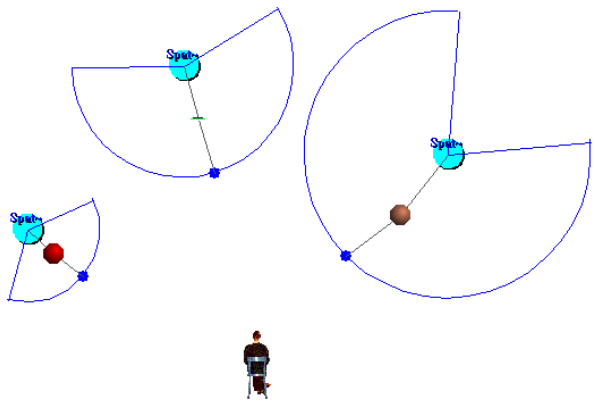


Figure 84 : exemple de trois relations différentes entre les sources sonores du Spat et leurs objets de directivité correspondants

Dans le deuxième cas, au centre sur la Figure 84, une contrainte de type « barre solide » est utilisée et permet d'assurer que la distance entre la source et l'objet directivité reste constant. C'est donc le paramètre de directivité décrivant l'angle d'ouverture de la source qui reste invariant dans cet exemple. Finalement, dans le dernier cas, une contrainte de type « écart angulaire constant » permet, contrairement à l'exemple précédent de figer l'orientation de la source par rapport à l'avatar et donc de manipuler la source sonore en conservant l'angle sous lequel elle voit l'avatar. Ces exemples de base peuvent bien sûr être améliorés, ou complexifié, par combinaison avec les autres contraintes de notre système.

Pour finir, cette expérience pose également le problème intéressant du contrôle d'un même jeu de paramètres simultanément par deux approches différentes. Le Spatialisateur IRCAM jongle constamment avec une approche physique de description de scène, rassemblant les paramètres géométriques de position et d'orientation des sources, et une approche perceptive. Les deux approches se défendent bien entendues, et ont toutes deux été incorporées au standard MPEG4, comme décrit dans la section 1.4.3, mais la manière de faire cohabiter ces deux approches de manière cohérente n'est jamais complètement abordée. Dans le cas du Spat, précisément, il existe une équivalence entre d'une part le paramètre perceptif dit de « présence » d'une source, et d'autre part la combinaison des paramètres physique de distance de la source à l'auditeur, et de « rolloff » précisant la loi d'éloignement à utiliser.

Ces deux paramètres « haut-niveau » agissent donc sur des éléments bas-niveau similaires dans la chaîne de traitement du signal et le contrôle de indépendant de chacun d'entre eux ne fait pas forcément sens, l'un annulant l'autre. C'est typiquement ce genre de cas de figure auquel notre système sait faire face : le mécanisme de contraintes faciliterait l'expression de relations entre les paramètres physiques et les paramètres perceptifs de la scène sonore et permettrait donc de contrôler celle-ci, de manière cohérente, indifféremment à l'aide de l'une ou de l'autre approche.

Troisième Partie
Expérimentation

L'évaluation de notre système n'est pas une opération simple. Pour parler d'évaluation, il nous faut d'une part repartir d'un objectif initial, et d'autre part une méthode permettant de décrire dans quelle mesure ce but a été atteint. Par souci d'objectivité, ces méthodes ont souvent recours à des phases de passation durant lesquelles l'avis d'un nombre important de sujet est recueilli et factorisé pour ensuite être analysé au moyen souvent de techniques statistiques.

Dans le cas de notre projet, il est difficile de répondre aux questions d'évaluation de manière abstraite, sans s'appuyer sur un domaine d'application précis. MusicSpace en met un certain nombre en évidence, qu'il s'agisse d'applications destinées au grand public, ou d'application réservées aux professionnel du mixage, ou encore d'applications plus expérimentales intéressant plus particulièrement les artistes.

Nous décrivons donc dans cette partie un ensemble de mises en situations de notre prototype, chacune se distinguant des autres soit par la technique de spatialisation employée, soit par le type document sonore utilisé, soit par le type de résultat souhaité. Si nous insistons sur cette phase d'expérimentation, c'est parce que c'est elle, enrichissante, qui permet d'envisager de nouvelles applications, de faire émerger de nouveaux cas particuliers, et donc de nouveaux problèmes à résoudre.

Dans un premier temps, au Chapitre 7, nous décrivons l'expérience du « trio de jazz » exemple pilote qui a permis d'établir les bases de notre système. Vient ensuite, au Chapitre 8, l'exemple « Ken Mouka », qui s'approche de la réalité en étant construit sur de la musique populaire existante ([Wes, 1996]) prêtée gracieusement par Sony Music pour le cadre de ces recherches.

Si les deux applications précédentes pouvaient avoir des portées orientées vers des utilisations grand public, le Chapitre 9 met en lumière une application plus marginale de notre système où nous montrons qu'il est possible de contrôler d'autres paramètres que ceux de la spatialisation et de mettre en œuvre le système de contraintes pour construire des effets originaux et intéressant. C'est dans cette partie d'avantage aux compositeurs où aux créateurs que nous nous adressons.

Pour fini, le Chapitre 10 décrit la phase d'expérimentation la plus importante puisqu'elle

concerne le travail de recherches autour de notre projet, effectué par Nicolas Deflache, ingénieur du son, dans le cadre de ses études. C'est donc en confiant notre prototype à un ingénieur du son que nous terminons cette partie sur l'expérimentation, et recueillons précieusement un premier tronçon d'évaluation.

Chapitre 7
JazzTrio.

Ce morceau, issu d'un thème de jazz de Rodgers & Hart intitulé « Falling in love with love » [Rodgers & Hart, 1938] et orchestré dans ce cas précis pour une formation de trio de jazz (piano, contrebasse et batterie) est un exemple simple permettant déjà de raisonner en termes de relations entre les instruments et donc de mettre directement en évidence les possibilités et l'intérêt des contraintes.

Figure 85 : configuration initiale du trio de jazz

La configuration de base de cet exemple (Figure 85) représente les instruments comme dans une formation de trio de jazz traditionnelle : les instruments sont « sur scène », devant l'auditeur, la batterie est au fond, la contrebasse à proximité de la batterie pour former la section rythmique, dont le piano, soliste, se détache en s'approchant de l'avant de la scène.

Comme il a été montré dans la section Chapitre 4, cette configuration initiale est fragile et peut rapidement être amenée à des situations de mixage incorrectes et incohérentes. Nous proposons donc, pour cet exemple, un ensemble de contraintes permettant de spécifier un minimum d'information relatif aux sources sonores de telle sorte que les manipulations de la scène restent toujours valables.

Dans un premier temps (Figure 86), une contrainte de type « rapport des distances constant » permet de lier la batterie à la contrebasse de manière à figer l'équilibre des intensités respectives de ces deux instruments et former ainsi la section rythmique de la formation instrumentale. Au contraire, une contrainte de type « produit des distances constant » reliant le piano à la batterie (et donc par « transitivité » à la contrebasse) permet de mettre en avant le rôle soliste du piano en le dégageant du reste de l'effectif. Finalement des contraintes de type « limite radiale » expriment la marge de manœuvre que l'utilisateur aura sur la distance du piano à l'avatar et permettent de faire en sorte que le piano ne soit jamais trop près, mais reste toujours audible,

en particulier en regard des deux autres instruments.

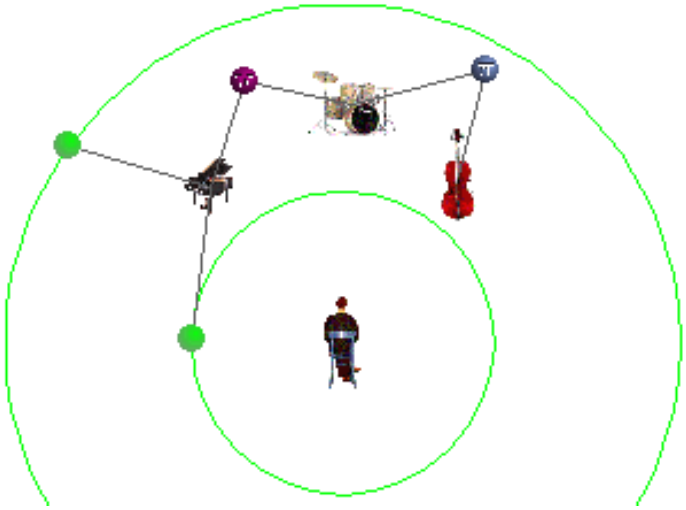


Figure 86 : trio de jazz et contraintes de distance

Dans un deuxième temps (Figure 87), des contraintes angulaires sont ajoutées de manière à affiner les relations entre les différents instruments : une contrainte de type « écart angulaire constant » permet de figer l'écart angulaire entre la contrebasse et la batterie (à une valeur relativement faible, d'environ 30 degrés) de sorte à associer leur positions spatiales et à renforcer la notion de section rythmique. Par ailleurs, deux contraintes de type « limite angulaire » permettent d'assurer que l'ensemble des instruments reste à l'avant par rapport à l'auditeur.

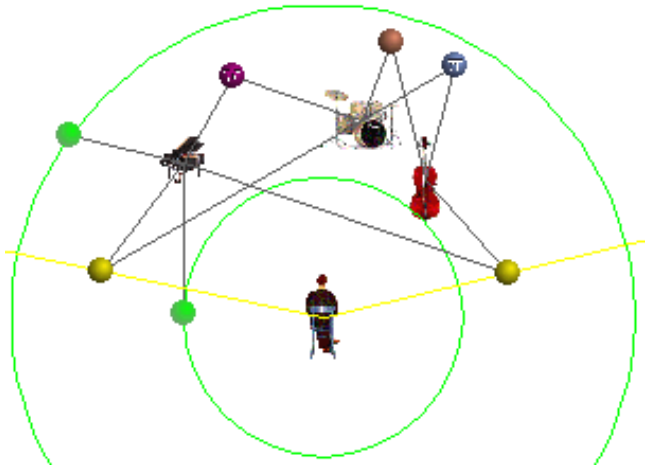


Figure 87 : trio de jazz, exemple complet

En résumé, le résultat est une scène sonore dans laquelle l'utilisateur peut ajuster le dosage entre la partie soliste et la section d'accompagnement : ces modifications s'effectuent au travers de l'ajustement de la distance des différents instruments au sein des marges autorisées par les contraintes de limite de distance. Par ailleurs, l'utilisateur peut modifier l'organisation spatiale des instruments toujours dans certaines limites, spécifiées cette fois ci par les contraintes de

limite angulaires : les sections « soliste » et « rythmique » peuvent êtres positionnées indépendamment l'une de l'autre, mais les instruments à l'intérieur de la section rythmique sont tenus, par une contrainte sur les écarts angulaires, à rester voisins.

Pour finir, cet exemple aurait pu être poussé plus loin, étendu à partir des considérations évoquées dans la section 6.1 au sujet de ce même exemple sonore, dans laquelle grâce au particularités de l'encodage MIDI la partie de piano était divisée en deux sources sonores représentant les parties de main gauche et de main droite. Le système de contraintes de MusicSpace aurait alors servi à établir des relations entre ces deux sources et les autres instrument de l'effectif, pour positionner la main gauche du piano quelque part entre la section d'accompagnement dont c'est le rôle et son appartenance à l'instrument soliste.

Chapitre 8
Wes / Ken Mouka

L'exemple de Wes, « Ken-Mouka » [Wes, 1996] nous est gracieusement prêté par Sony Music et est construit à partir des enregistrements originaux, multipistes, de la chanson. Le chanteur, Wes (Wes Madiko) associe dans cette chanson ses traditions Bantou (Cameroun) au savoir faire électronique de Michel Sanchez (« Deep Forest »). Cet enregistrement a été tout d'abord re-échantillonné, piste par piste. Les échantillons sonores obtenus ont ensuite été re-synchronisés avant d'être entrelacés au format WAV pour donner finalement un seul fichier audio utilisable dans MusicSpace.

Hormis le fait qu'il est l'un des rares exemples au format multipistes qu'il a été possible d'obtenir, nous avons retenu ce morceau de musique populaire parce qu'il se prête particulièrement bien à notre jeu et se laisse découvrir sous différentes facettes suivant les priorités de mixage choisies.

Nous en présentons ainsi tout d'abord trois différents arrangements, statiques, construits à l'aide de MusicSpace à partir des sources originales : « a capella », « techno » et « unplugged ». Ces versions diffèrent sensiblement sur le plan sonore et s'adressent presque à des publics différents. Dans un deuxième temps, nous présentons une version plus « dynamique » du morceau, faisant usage du mécanisme de « handle » et contraintes multiway présenté au paragraphe 5.2.2.

L'enregistrement est réparti sur 11 pistes différentes nommées arbitrairement :

Acoustic	Les différents instruments acoustiques utilisés (sanza, orgue à bouche, kora,…). Leur rôle est particulièrement au moment du refrain. Ils restent plus ou moins en arrière plan pendant les couplets.
Techno	Les éléments électroniques, essentiellement des sons synthétiques joués « pizzicato ».
FX1 FX2	Les effets sonores : balayages harmoniques par exemple, utilisés dans les phases de transition telles que couplet / refrain. Ces effets sont « stéréo » :
Drums	Les percussions.
Strings	Une section de cordes tenues, synthétiques, au rôle essentiellement harmonique.
Bass	La voix de guitare basse (électrique).
LeadVoice1 LeadVoice2	La voix principale du morceau, celle de Wes. Enregistrée de près sur deux canaux. L'enregistrement stéréophonique de proximité n'a pas tant d'importance au niveau spatial par rapport à un enregistrement monophonique :

	il vise surtout à conserver la présence dans le résultat sonore en particulier pour compenser d'éventuels mouvements du chanteur lors de l'enregistrement.
EnsembleVoice1 EnsembleVoice2	Les voix secondaires qui donnent la réplique à Wes. Elles sonnent comme un chœur d'enfants et sont enregistrées sur une paire de microphones stéréophonique.

Le premier arrangement de ces pistes proposé est la version « a capella » du morceau (Figure 88), style dans lequel ne sont conservées traditionnellement que les voix humaines. Dans notre exemple, toutes les pistes instrumentales sont donc mutées à l'exception de la guitare basse qui vient soutenir l'harmonie sans détruire le caractère essentiellement vocal du mixage. La voix principale est positionnée en avant au centre de l'image sonore, tandis que les voix secondaires apparaissent clairement sur les cotés mais restent très présentes. La basse est positionnée au centre également pour ne pas créer de déséquilibre mais légèrement en retrait.

Figure 88 : Wes / Ken Mouka, configuration "*a capella*"

Le terme « unplugged » (« débranché ») vient en opposition du terme « électrique » : dans cet exercice les instruments électriques sont débranchés, les guitares électriques remplacées par des guitares acoustiques,… L'artiste propose une version acoustique de ses chansons dans laquelle il ne recherche plus nécessairement la course aux décibels mais une relation avec le public plus intime et probablement une plus grande précision dans le jeu instrumental et la qualité acoustique.

Bien que les enregistrements originaux de Ken Mouka n'aient pas été conçus dans cette optique, nous proposons tout de même un arrangement des différentes sources sonores qui visent à imiter le style unplugged (Figure 89). Dans cette version, les instruments électriques sont donc mutés, incluant la source « drums », et à l'exception de la source « strings » qui, bien que de nature synthétique vise à imiter le jeu d'une section de cordes acoustique.

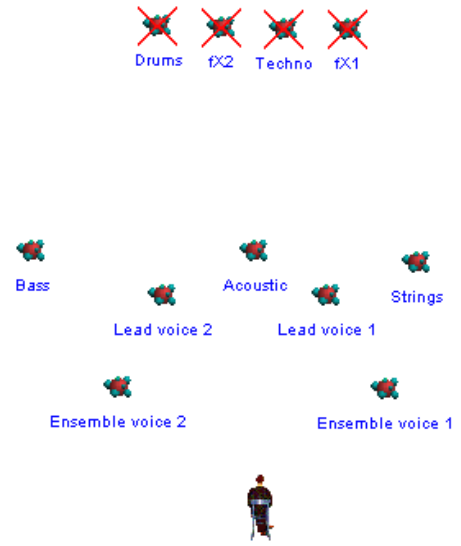


Figure 89 : Wes / Ken Mouka, configuration "unplugged"

Le troisième arrangement de sources proposé est nommé : « version techno » (Figure 90). Dans cet arrangement, l'accent est mis sur l'aspect rythmique et répétitif de l'accompagnement. Les voix, bien que nécessaires pour assurer un minimum de diversité sont positionnées derrière l'avatar et de manière assez distante, dans une zone, donc, où l'auditeur porte moins d'attention. Au contraire, les sources rythmiques et synthétiques sont disposées en face et à proximité de l'avatar.

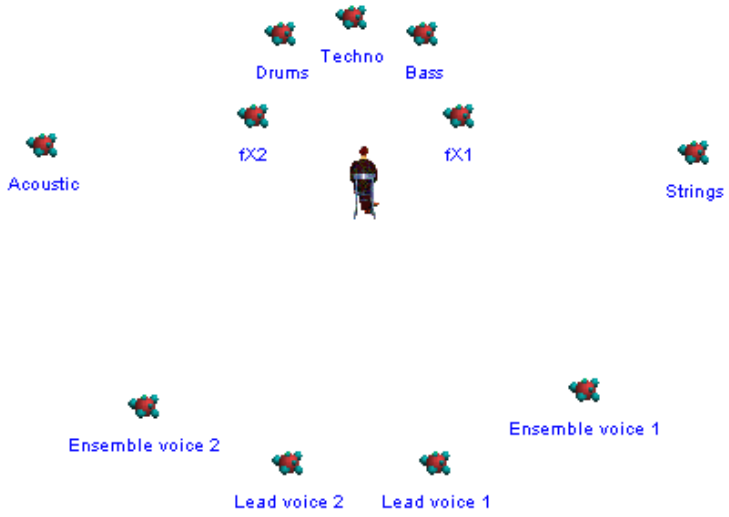


Figure 90 : Wes / Ken Mouka, version "techno"

Accéder à partir d'un même ensemble de pistes sonores, à différentes versions d'un morceau nous paraît déjà une idée attrayante. Le choix musical que fait un auditeur en achetant un disque pourrait être par exemple complété par celui d'un arrangement, une touche finale, qui lui correspond au mieux. Mais, mieux encore, rien n'interdit à cet arrangement d'évoluer dynamiquement. Nous proposons donc, toujours à partir des sources originales de Ken

Mouka, un arrangement dit « dynamique » permettant à l'auditeur d'ajuster lui-même le rendu final du mixage (Figure 91).

À la différence de l'exemple « Trio de Jazz » présenté dans la section précédente, le morceau Ken Mouka comporte un nombre important de sources sonores : celles-ci deviennent difficiles à gérer individuellement et nous proposons de faire usage du mécanisme de « handles » pour réduire le nombre d'objet à contrôler, et proposer des paramètres de contrôle plus intuitifs.

Nous faisons usage du mécanisme de contraintes de MusicSpace pour exprimer un certain nombre de particularités de la formation instrumentale à partir de relations entre les sources sonores. Celles-ci sont réparties entre quatre groupe : les voix humaines d'une part (voix lead et voix d'accompagnement), la section dite « acoustique » regroupant les cordes – bien que d'origine synthétiques – ainsi que sanza, cora,…, la section « électronique » regroupant les sons synthétiques et pistes d'effets sonores, et pour finir la section rythmique, composée de la basse et la batterie.

Un objet de type handle est construit pour chacune de ces sections de manière à permettre le contrôle simultané de l'ensemble des sources sonores qui les compose. Grâce au paradigme de contraintes « one way », l'utilisateur peut à sa guise agir individuellement sur une source – tant en distance qu'en orientation par rapport à l'avatar – pour établir un équilibre sonore à l'intérieur d'une section, ou sur les handles eux-mêmes afin d'établir l'équilibre général entre les sections.

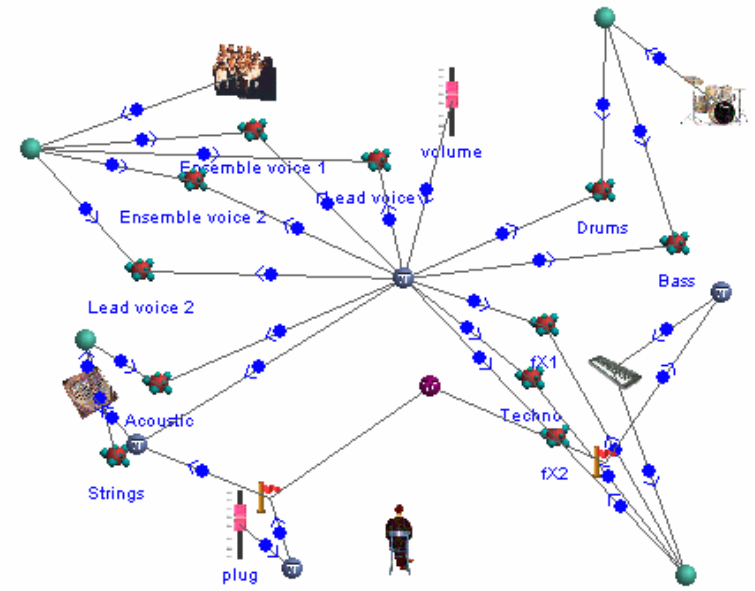


Figure 91 : Wes / Ken Mouka, configuration "dynamique"

Par ailleurs deux paramètres de contrôles supplémentaires ont été ajoutés : le premier, dénommé « volume » agit par l'intermédiaire d'une contrainte de type « rapport des distances deux à deux constants » de manière proportionnelle sur les distances à l'avatar de chacun des handles décrits précédemment, et donc indirectement (et toujours proportionnellement) sur les distances de chacune des sources sonores à l'avatar.

Le second, appellé « plug » permet de manipuler les handles « acoustics » et « electrique » de

manière antinomique : en déplaçant le contrôleur « plug » par rapport à l'avatar, nous favorisons donc soit les instruments acoustiques de l'effectif, soit les instruments électriques, mettant en évidence la possibilité de passer de manière progressive d'une version « unplugged » du morceau vers une version plus électronique.

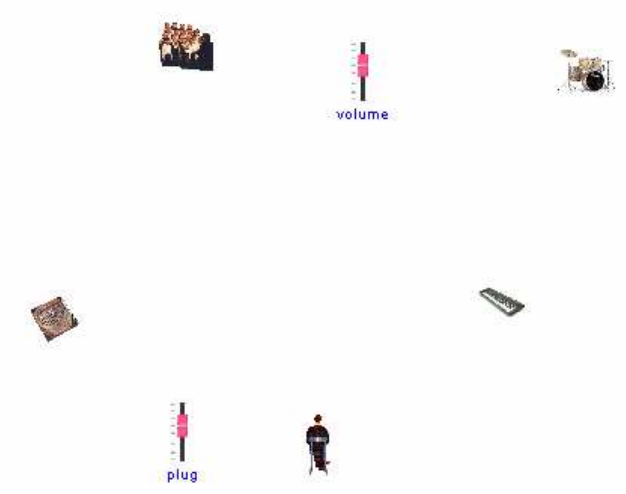


Figure 92 : Wes / Ken Mouka. Configuration "dynamique" en mode "LISTEN"

Au final, dans le mode d'utilisation courant de MusicSpace, le mode « LISTEN », les éléments bas niveau (sources sonores et contraintes) de la scène sont masqués pour ne laisser d'accès qu'aux éléments haut niveau que nous venons de décrire (Figure 92). La représentation graphique est ainsi largement simplifiée pour ne laisser apparaître que les contrôles que nous avons choisis :

- Quatre objets rassemblant intuitivement les sources sonores par groupe ou par fonction
- Deux « boutons » permettant d'agir d'une part sur le volume global et d'autre part sur l'équilibre entre les sources acoustiques et les sources synthétiques du morceau.

Chapitre 9
Filtrage

Cette section présente l'expérience inévitable du contrôle de paramètres qui ne sont pas forcément liés à la spatialisation. Ces paramètres ne possèdent donc plus nécessairement d'analogie avec une distance physique ou un écart angulaire par rapport à un auditeur. C'était déjà le cas avec la notion de handle présentée précédemment, dans laquelle simplement rapprocher le handle de l'avatar signifiait « plus de ce paramètre » et non forcément « plus proche, plus de signal direct et moins de signal réverbéré ».

Pour ce type d'expérience plusieurs objets graphiques ont été crées permettant le contrôle de paramètres MIDI arbitraires. Le premier, nommé « Generic MIDI Control Change Object » émet des valeurs de message de type « control change » en fonction de sa position. Sa fenêtre de propriété, représentée Figure 93 permet de voir comment les paramètres géométriques sont mis en regard des paramètres de contrôle. Les coordonnées de l'objet, en représentation polaire ou cartésienne, avec une mesure d'angle sur 360 degrés ou sur 180, avec des coordonnées cartésiennes en mode absolu (par rapport à la fenêtre de l'application) ou relatif (par rapport à la position de l'avatar) sont transcrites en valeur de paramètres contrôle change en fonction du choix de l'utilisateur.

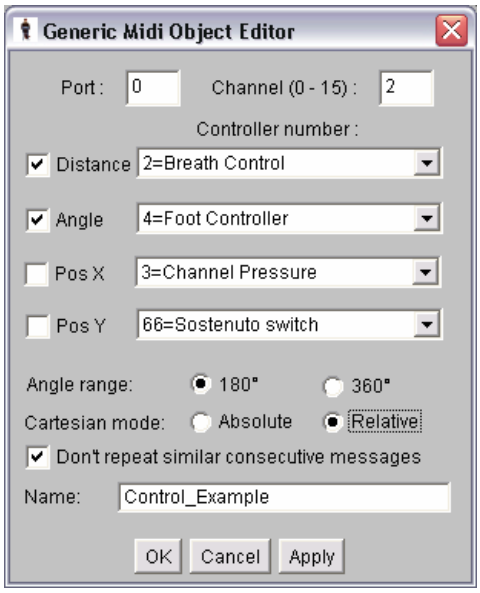


Figure 93 : vue de l'éditeur de l'objet « Generic MIDI Control Change Object »

Celui-ci peut sélectionner les grandeurs géométriques (distance, angle, position X, position Y) à transcrire en valeur de contrôle et décider quel numéro de paramètre doit être utilisé pour chacune d'entre elle au moyen d'une liste sous forme de menu déroulant rappelant les

différents numéros de contrôleurs retenus par la norme MIDI et le standard General MIDI.

La seconde classe d'objet conçu pour ce type d'application sont dénomés « Generic MIDI object » et concernent initialement le contrôle des paramètres de la console Yamaha O2R tels que les paramètres de filtrage. La console Yamaha O2R est entièrement contrôlable à distance par l'intermédiaire de communication MIDI. Mais le constructeur a préféré, au lieu de monopoliser des canaux MIDI pour implémenter ces contrôles au sein de messages courts, utiliser la communication par « système exclusif » permettant d'adresser n'importe quelle valeur de la console.

L'interface de contrôle de ces objets est représentée en Figure 94. De manière similaire à l'éditeur d'objet présenté précédemment, elle permet d'effectuer des correspondances entre ses valeurs de distance et de position angulaire par rapport à l'avatar, mais le simple choix d'un message control change est remplacé par une zone de texte ou le message à transmettre est décrit sous la forme d'une suite d'octets.

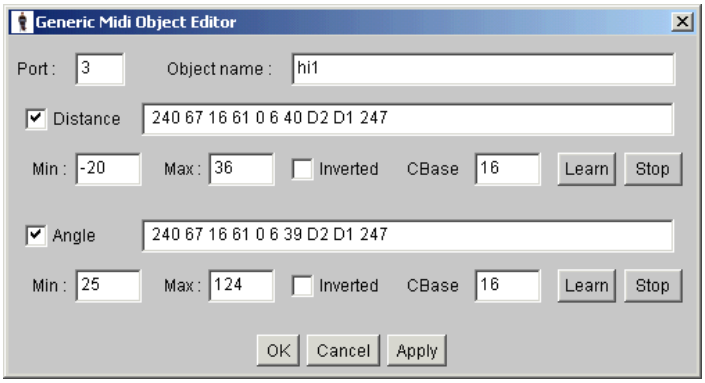


Figure 94 : Editeur de paramètres des « Generic Midi Object ».

Dans l'exemple présenté en Figure 94, la distance de l'objet par rapport à l'avatar est transcrite en message « 240 67 16 61 0 6 40 D2 D1 247 ». Le premier octet, 240 signifie le début d'un message de type « Système Exclusif », les valeurs suivantes « 67 16 » permettent d'identifier la console Yahama O2R par son code de constructeur et de modèle. Puis vient « 61 0 6 40 » déterminent la nature du message (changement de paramètre) ainsi que la valeur de ce paramètre. « D2 D1 » représente la partie variable de ce paramètres : les autres champs de cette interface nous enseignent que cette valeur doit varier entre -20 et 36, et que la base numérique utilisée est 16. Ainsi pour transmettre la valeur 30 par exemple, nous opérons la décomposition en base 16 : 30 = 16 * 1 + 14 et nous en déduisons D2 = 1, D1 = 14.

Ces paramètres sont fastidieux à entrer dans la machine manuellement. Par ailleurs, il est nécessaire de posséder une certaine expertise pour pouvoir les formuler, en particulier à partir des tables d'implémentations MIDI, diagrammes obscurs, généralement situés en annexe des manuels d'utilisation des appareils… Pour palier ce défaut, nous proposons une solution qui consiste à inférer ces paramètres automatiquement, à partir des informations générées par la console lorsqu'on la manipule. Cette particularité est entièrement décrite en annexe de ce document, dans la section « apprentissage des fonctions MIDI ».

Nous présentons, en Figure 95, un exemple d'utilisation de ces objets de contrôle, dans lequel nous proposons à partir du contrôle de filtres, une forme de balance complètement originale

entre deux pistes de la console de mixage.

Dans cet exemple, nos objets midi permettent de contrôler des filtres de la console de mixage. Ces filtres sont de type « passe bande », ajustables en amplitude ainsi qu'en fréquence. Nous avons choisi ici arbitrairement de contrôler la fréquence de ces filtres à partir de la position de l'objet par rapport à l'avatar : une position complètement à gauche correspond à un e fréquence de 20 Hz, tandis qu'une position complètement à droite représente l'extrême aigu, à 20 kHz. Par ailleurs, le réglage en amplitude, dépend de la distance de l'objet à l'avatar, variant de -10 dB au plus loin jusque linéairement +18 dB au plus près.

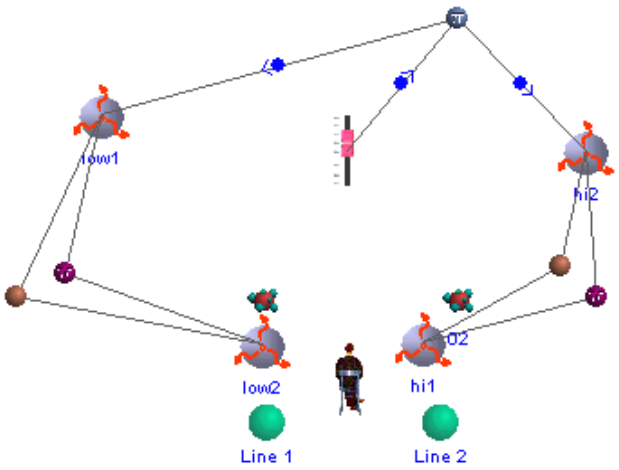


Figure 95 : Quatre filtres en opposition sur deux voies de la console de mixage.

Nous pilotons ici quatre filtres répartis sur deux voies (line 1 et line2) de la console de mixage : low1 et hi1 pour la première, low2 et hi2 pour la seconde. Des contraintes de produit des distance et écarts angulaires constant relient ces filtres deux à deux, low1 et low2 d'une part, et hi1 et h2 d'autre part, de telle sorte que les filtres graves (low1 et low2) et les filtres aigus (hi1 et hi2) aient la même fréquence mais soient en opposition au niveau du gain.

Ce système permet tout d'abord d'ajuster les fréquences des filtres en déplaçant ceux-ci autour de l'avatar ainsi que les rapports de gain aux conditions initiales, entre les différents filtres.

Par ailleurs, une fois les ajustements préliminaires effectués, un objet de type « handle » permet de contrôler l'ensemble de la scène et produit lorsqu'on le manipule, un effet de balance intéressant autant qu'original entre les pistes 1 et 2 de la console dans le résultat sonore, en privilégiant soit les fréquences graves de la piste 1 avec les fréquences aigues de la piste 2, soit, au contraire, les fréquences aigues de la piste 1 avec les fréquences grave de la piste 2.

Chapitre 10
Mise en situation réelle

Dans le cadre de sa « formation supérieure aux métiers du son » suivie au Conservatoire National Supérieur de Musique de Paris, Nicolas Deflache a largement contribué à l'évaluation de notre système en y consacrant son projet de recherche de fin d'études [Deflache, 2001]. Son travail est enrichissant à plusieurs niveaux et a permis, entre autres, l'évaluation de notre prototype dans des conditions réelles d'utilisation. Le but de ce travail est la « réalisation d'un mixage de pop-music interactif », c'est-à-dire de produire un système permettant à un auditeur d'intervenir lors de l'écoute d'un enregistrement sonore de sorte à l'adapter à ses préférences. Essentiellement, cette étude propose d'une part de construire différents « points d'écoute » d'une scène sonore, et d'autre part de mettre en œuvre des règles de transitions permettant de passer progressivement d'un point d'écoute à un autre.

L'étude classifie les paramètres musicaux de la pop-music en deux catégories : les paramètres « musicaux mobiles », comprenant l'arrangement, l'instrumentation et l'interprétation, et les paramètres « sonores mobiles » qui décrivent les transformations des caractéristiques auditives de l'image sonore, sans toucher au contenu musical. Ces derniers sont par exemple la largeur apparente de l'image sonore, la perception de la qualité acoustique de l'environnement, les transformations du timbre d'un instrument lié à la distance à laquelle il est capté, ou encore le mouvement des sources autour de l'auditeur.

Ces paramètres ne sont jamais indépendants les uns des autres et l'auteur propose par exemple un contrôle global mettant en jeu tous ces paramètres simultanément et permettant de faire varier progressivement le mixage d'un style intimiste (sources peu écartées, acoustique matte, arrangement doux, sources présentes,…) vers un style plus extraverti où l'image stéréophonique est large, l'acoustique réverbérante, les sources relativement distantes et l'arrangement plus dense.

La préoccupation principale mise en évidence est donc la cohérence du mixage : quels sont techniquement les relations à vérifier pour que l'image sonore reste toujours cohérente ? L'auteur s'appuie en particulier sur les résultats des recherches de Bergame Périaux [Périaux, 2000], concernant la « cohérence de l'image sonore en mixage variétés 5.1 ». Les critères qui apparaissent importants à cet effet sont entre autres l'équilibre spectral, l'homogénéité, la « balance », et la qualité de la définition et de l'intelligibilité. Evidemment, ceux-ci risquent d'être remis en cause à chaque modification des paramètres de spatialisation, d'orchestration ou d'arrangement.

Dans la partie pratique de cette étude, MusicSpace est utilisé pour le contrôle d'une console de mixage Yamaha O3D, déjà prévue pour le mixage surround au format 5.1. Des objets spécifiques pour cet appareil ont été ajoutés au système de manière à contrôler indépendamment le contrôle de gain sur chaque tranche et les paramètres de panoramique

surround qui curieusement s'expriment en coordonnées cartésiennes (une valeur en abscisse et une valeur en ordonnées) et semblent respecter une « distance constante » lorsque les paramètres décrivent un carré autour de l'avatar.

Le contenu sonore, « Gena » est une chanson pop-rock écrite par Fabrice Ducognon et Julien Perraudeau, enregistrée spécialement pour l'étude. La Figure 96 donne un aperçu de notre système ainsi que des quatre paramètres « globaux » proposés (« Stéréo/multicanal », « Réverbération », « Arrangement » et « Densité ») dans le cadre de cet exemple sonore. Derrière ces paramètres haut-niveau se cachent évidement les sources sonores initiales, ainsi qu'un ensemble de contraintes permettant de mettre en évidence la relation sur le mixage propre à chacun des paramètres.

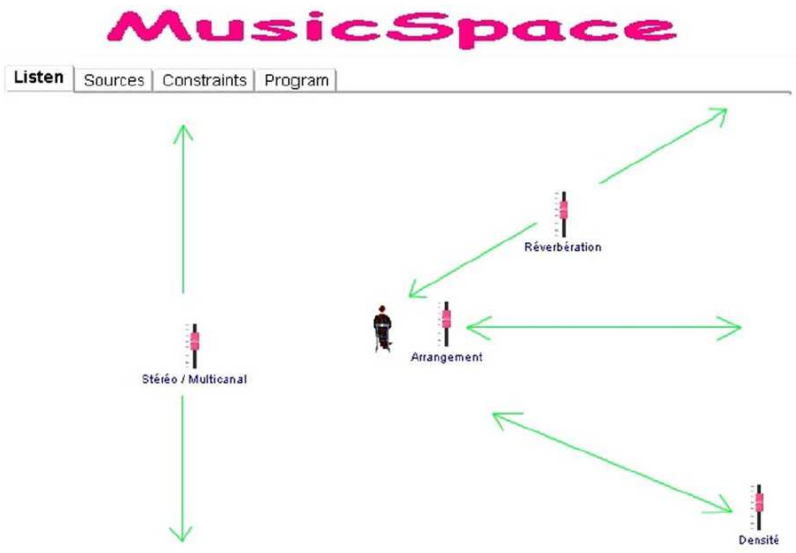


Figure 96 : proposition de quatre paramètres globaux

Le paramètre « Réverbération » agit sur la durée et le niveau de réverbération du résultat sonore en contrôlant les paramètres des processeurs d'effet utilisés (Lexicon 300, Eventide H3000 et TC System M6000). Les variations de ce paramètre permettent de passer progressivement d'une acoustique sèche (élargissement et légère coloration des voix principales et solistes à l'aide de réflexions précoces) à une acoustique plus réverbérée (qui combine une réverbération sourde de 3,8 secondes jouant le rôle de pré-délai, et une réverbération multicanale plus longue).

L'enregistrement initial a été conçu de sorte que deux arrangements différents soient possibles pour ce morceau : le contrôleur « Arrangement » permet donc précisément de basculer d'un arrangement « acoustique » (rythmique ternaire à décomposition binaire) vers un arrangement « électrique » (rythmique binaire à décomposition ternaire) en agissant essentiellement sur l'orchestration de l'accompagnement. L'ensemble des sources sonore est ainsi divisé en trois groupes :

- Sources communes aux deux arrangements : Voix lead, Guitare lead, Accordéon et trombone
- Sources spécifiques à l'arrangement acoustique : Contrebasse, Batterie (jouée aux

balais) et deux guitares acoustiques

- Sources spécifiques à l'arrangement électrique : Basse synthétique, Boite à rythme et Fender Rhodes.

A titre d'exemple, la Figure 97 donne un aperçu du contrôleur « Arrangement » et des sources qu'il pilote :

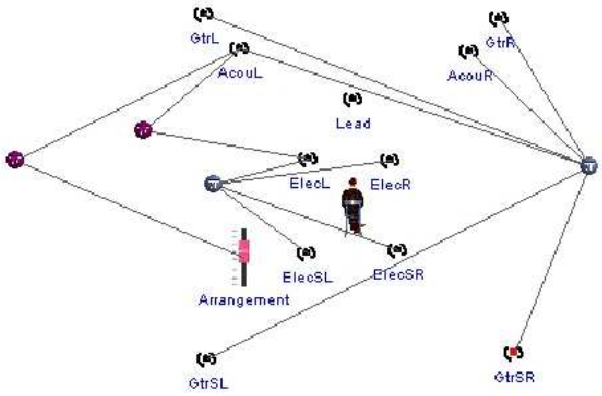


Figure 97 : le contrôleur "Arrangement"

Le paramètre de « Densité » concerne le contrôle de sources sonores dont le rôle est de renforcer l'écriture dans les refrains et les parties instrumentales. Ces sources sont les Basses de l'accordéon, et certaines percussions. Il est question également de la guitare rythmique dans le cas d'un arrangement acoustique et du Fender Rhodes son saturé et bruits de surface vinyle pour l'arrangement électrique. Le handle « Densité » permet donc de faire varier la richesse de l'orchestration et ce de manière adaptée à l'arrangement choisi (cf Figure 98).

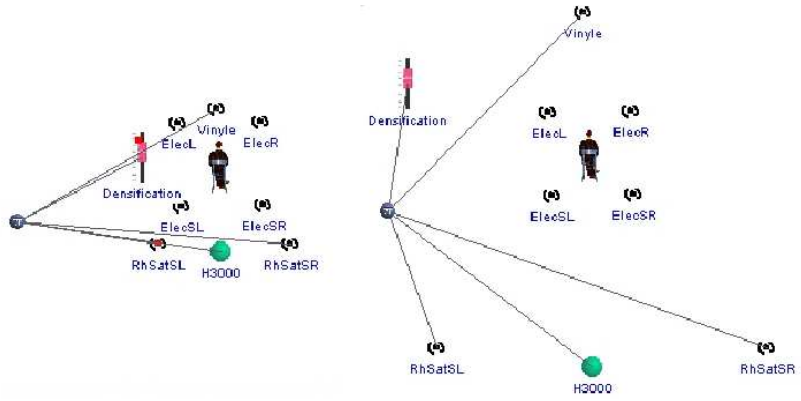


Figure 98 : contrôle de la "densité" dans le cas de l'arrangement "électrique"

Finalement, le contrôleur « Stéréo / Multicanal » permet de passer progressivement d'un résultat sonore stéréophonique (deux haut-parleurs gauche et droit situés devant l'auditeur) à une configuration multicanal de type 5.1 ou l'auditeur est au centre d'une dispositif de

reproduction sonore.

Les commentaires et critiques relatifs à notre système sont nombreux : sa spécificité à permettre le contrôle simultané d'un ensemble de paramètres a été immédiatement ressentie dans un panel d'ingénieurs et futur ingénieurs du son. Par ailleurs si les objectifs formulés à l'origine de cette étude (la construction de différents point de vue sur le mixage d'un ensemble de sources sonore et la mise en œuvre de méthode de transition entre ces points de vue) ont été atteints, Nicolas Deflache décrit certaines lacunes de notre système liées à la précision des réglages, pour lesquels un écart d'un décibel peut être suffisant pour détruire la cohérence du résultat sonore, mettant en évidence la fragilité du mixage sonore. L'auteur fait également état d'une distorsion de la représentation (adaptée à la quadriphonie) par rapport au système de diffusion utilisé, le surround 5.1. Pour finir, il semble que les contraintes dans notre système agissant sur la perception spatiales des sources sonores soient difficilement utilisables dans ce type d'exercice et que des contrôles supplémentaires agissant sur le timbre des sources sonores seraient intéressants afin de compenser les variations produites lors du contrôle de panoramique.

Conclusion

Nous nous sommes posés, dans cette étude, le problème du contrôle de la spatialisation : quelle interface de contrôle serait la mieux adaptée pour tirer entièrement partie des outils de spatialisation existants ? Comment remplacer les paramètres bas-niveau liés – voire dépendants – à la technologie de spatialisation employée par des paramètres haut-niveau compatibles avec les termes du discours musical des compositeurs ? Quelle interface proposer à un utilisateur non expérimenté pour qu'il puisse agir adapter le rendu sonore d'un morceau de musique tout en étant sûr que quelles que soient ses actions, le résultat sera toujours un « bon » mixage ? Toutes ces questions s'articulent autour du problème central qui consiste à représenter et maintenir les relations qui existent entre les différentes sources d'une scène sonore, gage de cohérence dans le résultat sonore.

Nous avons proposé de répondre à ce problème en faisant appel à la programmation par contraintes afin de fournir d'une part les moyens de modéliser ces relations entre les différentes sources sonores, et d'autre part des méthodes de calcul adaptées pour assurer que ces relations sont toujours vérifiées. Nous avons concentré nos efforts sur les paramètres de positionnement des sources sonores, à leurs coordonnées relatives par rapport à la position du point d'écoute, dans un plan horizontal : ces paramètres se prêtent bien au jeu des contraintes et les relations à maintenir entre les sources sonores s'expriment comme des relations géométriques dans un plan.

Différents modèles de moteurs de résolution de contraintes ont été présentés, chacun présentant ses particularités en termes de type de problème pris en compte et de complexité : cette démarche a mené à la conception d'un modèle ad hoc, inspiré des travaux existant. Cet algorithme, certes incomplet, permet tout au moins de tenir compte de l'ensemble des contraintes nécessaires à notre projet, un ensemble hétéroclyte où la présence de cycles, de non-linéarités, et de contraintes non fonctionnelles rendait le cadre général complexe à mettre en œuvre et incapable de répondre aux exigences du temps-réel.

Ces idées ont été intégrées dans un prototype, nommé MusicSpace, une interface de contrôle de la spatialisation, proposant une représentation de la scène sonore accompagnée d'un système de contraintes permettant d'établir graphiquement et simplement les relations intrinsèques et d'assurer que ces relations sont toujours vérifiées en « adaptant » au besoin les actions d'un utilisateur.

Dès ses premières versions, notre projet a suscité beaucoup d'intérêt : MusicSpace a fait l'objet de nombreuses présentations en conférences (Audio Engineering Society et International Computer Music Conference), de présentations invitées lors des Journées de l'Audition à l'université Paris XII (2000) ou lors de la deuxième édition de SynWorld à Vienne (1999). MusicSpace a été récompensé au Concours International de Logiciels Musicaux organisé par l'Institut International de Musique Electroacoustique de Bourges et distribué dans le cadre du Forum IRCAM et de la distribution du système Midishare de GRAME. MusicSpace et OpenMusic ont également été récompensés du prix SFIM du meilleur papier étudiant pour « OpensSpace » ([Delerue & Agon, 1999]) remis par la Société Française d'Informatique Musicale. Finalement, MusicSpace a fait l'objet de deux brevets industriels accompagnés de leurs extensions internationales.

Le succès qui a accueilli notre projet montre bien à quel point le problème qu'il soulève est important et n'avait à ce jour quasiment jamais rencontré de proposition satisfaisante ou fiable. La proposition que nous faisons avec MusicSpace devrait encore se préciser et s'affirmer à mesure que se poursuivent les occasions d'utilisation et d'expérimentation. De ces expériences devrait émerger la part de notre système qui reste éventuellement à améliorer. L'objectif, à terme, est de spécifier au mieux un ensemble de contraintes suffisament générique et stable ainsi qu'un cahier des charges précis pour l'algorithme de résolution correspondant afin de de se tourner vers la phase de standardisation du projet et d'incorporer les concepts mis en avance dans MusicSpace au sein des standards de distribution de documents multimedia.

Annexes & Compléments Techniques

« Parser » les fichiers MIDI

Nous avons décrit dans la section 6.1 les aspects de notre travail relatifs au format MIDI : les avantages et inconvénients que présentait cette situation de mixage. Nous ajoutons ici que la musique au format MIDI présente l'avantage de se communiquer très aisément par son aspect compact et d'être communément utilisée, en arrière plan de sites internet par exemple. Mais avant d'envisager la spatialisation d'un morceau MIDI, la lecture « intelligente » du fichier qui en donne la description – sous forme de messages de commande – a été une étape fondamentale que nous décrivons ici.

MusicSpace représente graphiquement la scène sonore sous la forme d'une organisation guidée par la notion de « source sonore ». La représentation interne des données musicales doit être similaire, et c'est cette représentation qu'il est nécessaire de construire à partir des informations contenues dans le fichier.

La norme MIDI spécifie deux formats de fichier pour la sauvegarde des données : les formats « MIDIFILE 0 » et « MIDIFILE 1 » : ce sont ces fichiers qui nous sont donnés à lires lorsque l'on charge un morceau en mémoire. Le format MIDIFILE 0, le plus ancien, ne possède pas la notion de piste : tous les événements temporels sont regroupés ensemble et la structure est donc à réinventer lors de la lecture. Le format MIDIFILE 1, version améliorée du précédent apporte cette information supplémentaire et permet de grouper les événements temporels dans des structures que l'on saura restituer à la prochaine utilisation. Malheureusement la manière donc sont utilisée ces pistes n'est pas standardisée et dépend essentiellement du logiciel qui les a construites. Dans la plupart du temps ce logiciel est un séquenceur. Bien qu'il dispose normalement d'un format de sauvegarde dont il est propriétaire qui lui permet de stocker non seulement le morceau de musique mais aussi l'état de l'ensemble des paramètres de l'application, il est également capable pour des raisons de compatibilité avec les logiciels de reproduction de sauvegarder ces données au format MIDI. La structuration des événements MIDI sous forme de piste n'est alors pas destinée au « player » qui généralement ne saurait qu'en faire, mais est mise à contribution pour refléter l'organisation interne des pistes du séquenceur, qui ne reflète pas forcément une organisation sous forme de « pistes

instrumentales » mais est liée à la manière dont le concepteur a effectué son travail : les informations peuvent d'une piste instrumentale peuvent être distribuées sur plusieurs pistes MIDI, et de la même manière une piste MIDI peut contenir des informations appartenant à plusieurs pistes instrumentales.

La notion de piste contenue dans le format MIDIFILE 1 n'est donc pas fiable, et c'est à partir des événements MIDI dits de « changement de programme » que nous allons réorganiser l'information de manière cohérente, correspondant au mieux à la notion de source sonore nécessaire dans notre projet. Les messages de changement de programme indiquent lors de la reproduction quel instrument utiliser (quel instrument synthétiser) pour un canal MIDI donné.

Après avoir reçu un tel message, par exemple un numéro de programme 1 (piano) sur le canal 1, le synthétiseur joue toutes les notes reçues sur ce même canal, en imitant le son du piano jusqu'à ce qu'un autre message de changement de programme lui indique de faire autrement.

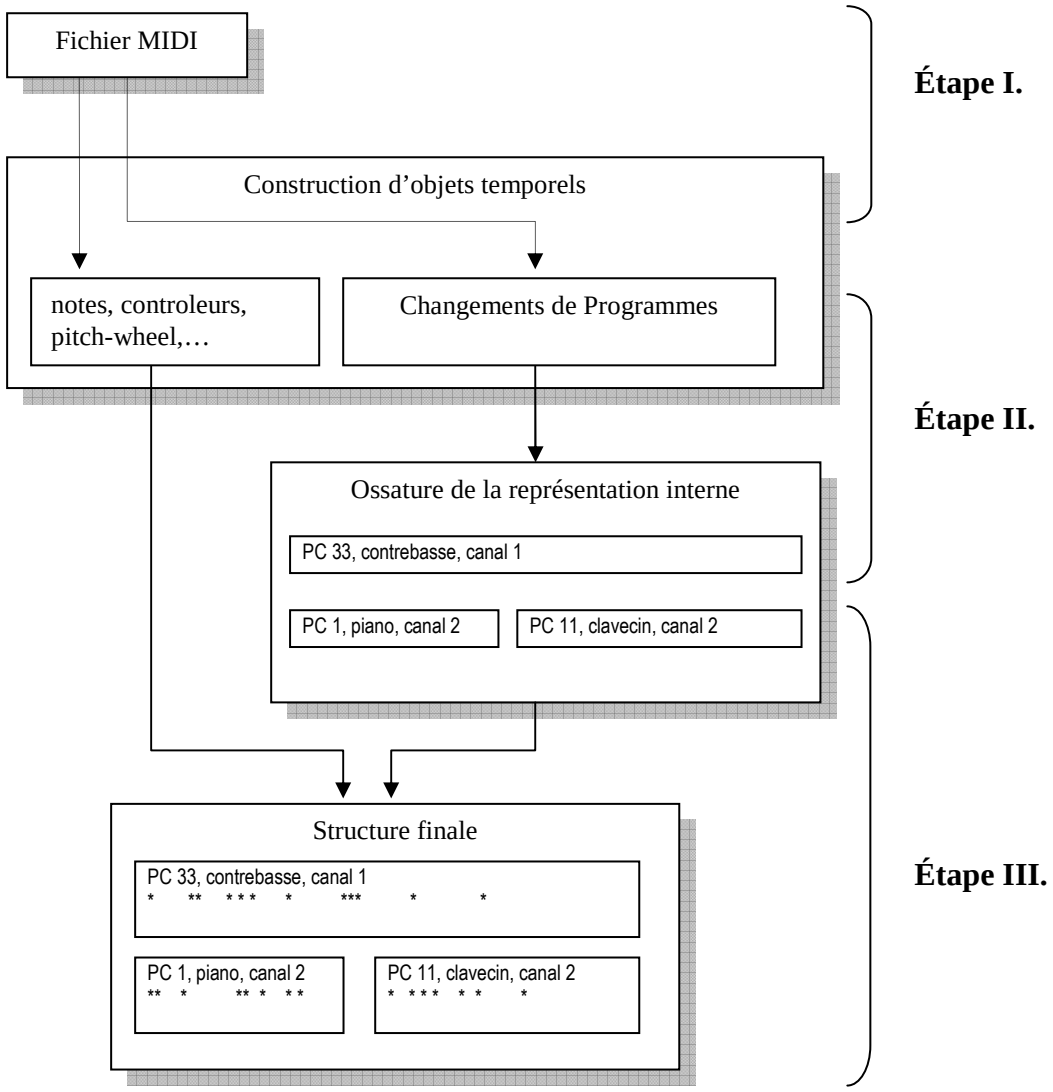


Figure 99 : étapes de l'analyse des fichiers MIDI

Nous avons, dans MusicSpace pris le parti de construire une source sonore par couple de numéro de programme et canal midi utilisé. Un même canal midi peut donc générer plusieurs sources sonores (qui joueraient en alternance). Par ailleurs deux canaux MIDI utilisant des numéros de programme identiques sont considérés comme des sources différentes. Ce choix possède l'avantage de permettre de rendre compte par exemple de deux flûtes jouant ensemble des parties différentes. En revanche, si les voix d'un instrument polyphonique ont été enregistrées sur des canaux MIDI différents (les parties de main gauche et main droite d'un piano par exemple), MusicSpace ne saura les regrouper pour reconstruire l'instrument original. Le choix dual aurait consisté à construire les pistes directement à partir des informations de changement de programme. Cette solution aurait permis de reconstituer le piano mentionné précédemment mais aurait fait l'amalgame entre les deux parties de flûte…

La Figure 99 décrit les trois étapes de notre algorithme d'analyse des fichiers MIDI. Tout d'abord (Etape I.) le fichier est parcouru dans son intégralité pour séparer d'une part les messages de changement de programme et d'autre part le reste des messages à savoir les notes et autres messages liés à l'interprétation. Dans la deuxième étape du procédé, une « ossature » de la représentation interne est construite, en se basant sur les messages de changement de programme. Dans notre exemple, trois sources sonores sont crées correspondant à une contrebasse jouant sur le canal MIDI 1, et un piano ainsi qu'un clavecin jouant l'un après l'autre sur le canal MIDI numéro 2. Pour finir, cette ossature est remplie avec les événements MIDI – essentiellement des notes - mis de coté initialement, à l'aide de leurs informations temporelles. Du fait que la spatialisation MIDI (voir section 6.1) est réalisée au moyen de messages de contrôle (control-change) pour les contrôleurs MIDI 7 (volume) et 10 (panoramique), ces événements – s'il en existe dans le fichier MIDI – ne sont pas conservés lors du remplissage des structures temporelles, pour laisser libre champ au spatialisateur.

Architecture(s) du séquenceur

Le problème du séquencement d'un morceau de musique consiste à faire jouer par un périphérique MIDI l'ensemble des notes de ce morceau, au moment où elles sont attendues. Si dans MusicSpace nous avons préféré construire nous même notre propre séquenceur plutôt que d'utiliser des systèmes déjà existant comme en particulier le séquenceur proposé par Grame dans le package Midishare (voir [Orlarey & Lequay, 1989]), c'est pour conserver le contrôle jusqu'au dernier moment sur ce qui doit être joué, et la manière dont cela doit être joué. Nous décrivons dans cette section l'architecture du séquenceur MIDI interne à MusicSpace, en détaillant les différents choix qui ont été faits lors de sa réalisation.

Typiquement, dans le problème de séquencement, la donnée est une liste d'événements temporels datés, ordonnée chronologiquement. Le plus tard la note jouée est confiée au système, le plus longtemps il est possible d'en garder le contrôle et éventuellement d'en modifier la hauteur, l'intensité ou le timbre. Une fois l'événement sonore transmis au système d'exploitation, il n'est plus possible de revenir en arrière, de le modifier ou de l'annuler ([Hanappe, 2000]). Le problème consiste donc à déterminer une architecture de gestion de tâche permettant de conserver le maximum de réactivité du système, typiquement en envoyant l'événement le plus tard possible au système. Nous proposons deux architectures envisageables pour la gestion de la tâche de séquencement.

Dans la première proposition (Figure 100) la tâche qui gère l'envoi temporel des événements sonores au système d'exploitation se réveille lorsqu'il y a des événements à jouer : dans notre exemple, la tâche se réveille en t_1 (1) puis transmet au système d'exploitation les événements à jouer à cette date au temps $t_1 + t_L$ (2) où t_L est la latence du séquenceur et finalement se rendort (3) jusque la date du prochain événement c'est-à-dire dans notre cas pour une durée de $t_2 - t_1$ millisecondes.

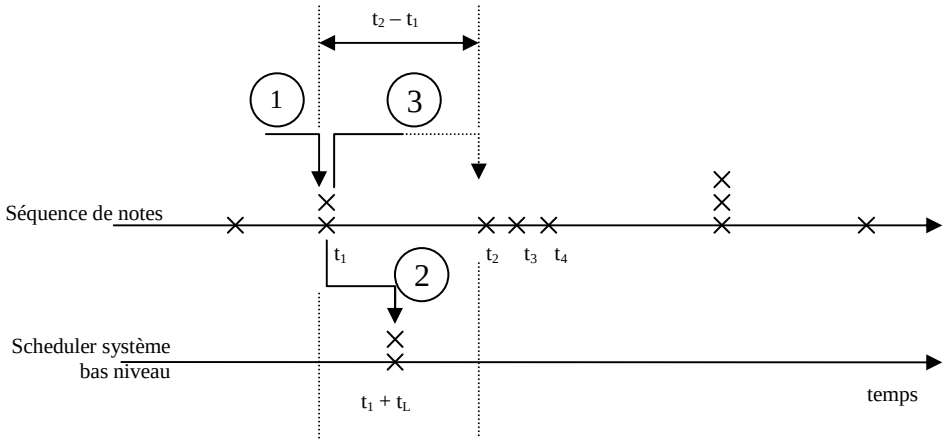


Figure 100 : Première proposition de gestion de tâche du séquenceur.
Le système se réveille à la date du prochain événement.

La latence du séquenceur apparaît de manière identique dans les différentes propositions des

tâches de séquencement : elle représente le délai supplémentaire que l'on s'autorise pour s'assurer que le système d'exploitation sera bien capable de jouer cette note au temps souhaité. Typiquement ce temps de latence est de l'ordre de 30 millisecondes.

Dans la deuxième proposition (Figure 101), la tâche de gestion temporelle se réveille périodiquement toutes les Δt millisecondes et va transmettre au système d'exploitation tous les événements sonores qui interviennent dans cet intervalle de temps. Dans notre exemple la tâche se réveille (1) à l'instant t_0. Elle transmet alors (2) au système tous les événements sonores dont la date est comprise entre t_0 et $(t_0 + \Delta t)$ avec la même latence t_L que dans le cas précédent. Finalement la tâche s'endort (3) pour une durée Δt, de sorte à se réveiller à l'instant $(t_0 + \Delta t)$. En pratique la durée de sommeil de la tâche est $(t_0 + \Delta t - \text{time}())$ où time() est la fonction qui renvoie l'heure actuelle, de sorte à prendre en compte le temps qu'a pris la transmission des événement sonores au système.

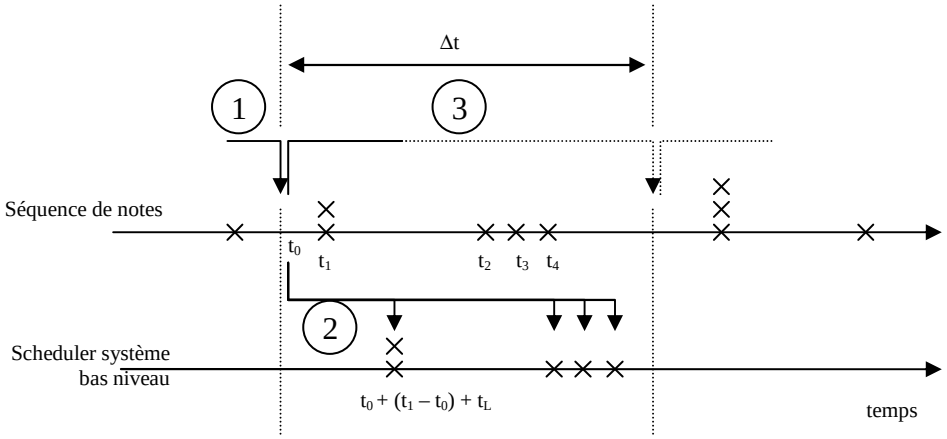


Figure 101 : deuxième proposition de gestion de tâche du séquenceur.
Le système se réveille à la date du prochain événement.

Chacun des deux cas présenté possède ses avantages et ses inconvénients. Dans le premier cas, la tâche de séquencement a l'avantage de ne se réveiller que lorsque cela est nécessaire. Cela permet de minimiser le coût en CPU entraîné par le changement de tâches. Cela autorise également une réactivité du système maximum puisque chaque note n'est envoyée au système d'exploitation qu'au dernier moment (au temps de latence près). En revanche, cette tâche est difficile à contrôler en cas de silence : la tâche s'endort pour une durée importante et il devient difficile d'arrêter ou mettre en pause le séquenceur de façon précise (Figure 102).

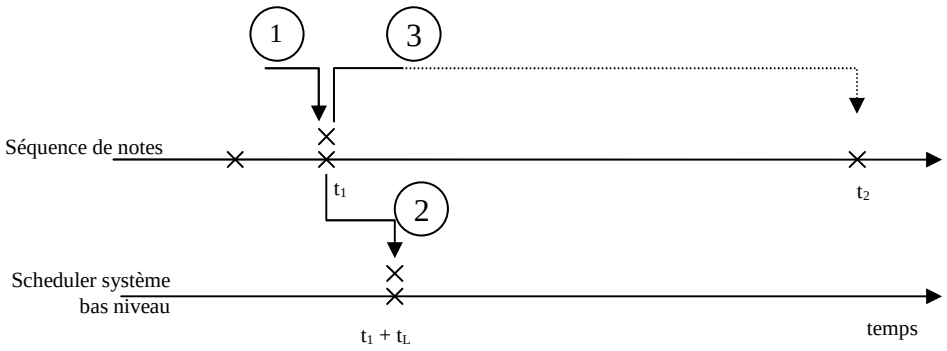


Figure 102 : une tâche peut rester inactive pendant une durée importante dans le

Dans le deuxième cas, la tâche de séquencement effectue des réveils inconditionnels périodiquement toutes les Δt millisecondes même si aucune note n'est à jouer (voir Figure 103).

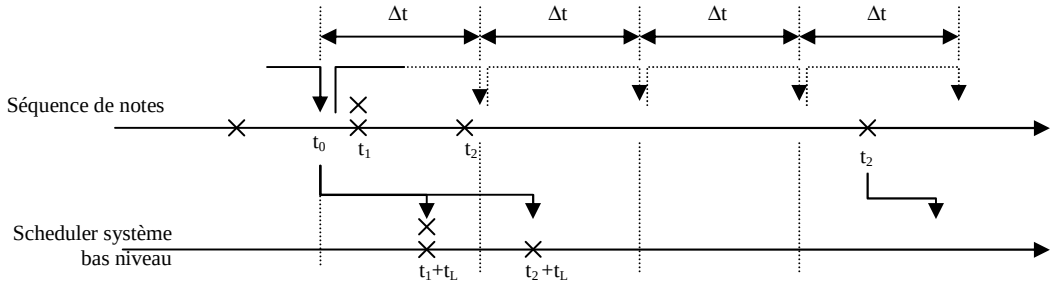


Figure 103 : lors de réveils périodiques, la tâche de séquencement peut éventuellement s'activer alors qu'aucune note n'est à jouer.

Le deuxième problème présente également un autre désavantage et réduisant la réactivité du séquenceur avec une latence supplémentaire pouvant atteindre les Δt millisecondes. Par exemple, l'événement sonore du temps t_2 sur la Figure 103 est programmé au moment où la tache se réveille c'est-à-dire en t_0. A partir de ce moment là, il n'est plus possible d'agir sur l'événement (le transformer, ou l'annuler). Le temps entre lesquels un événement sonore est programmé et celui ou il va être joué peut donc atteindre dans ce cas $\Delta t + t_L$.

C'est pourtant la deuxième solution que nous avons retenue pour notre système, car plus simple à mettre en œuvre. Une valeur adaptée a été choisie pour Δt (de l'ordre de 100ms) de sorte à éviter des réveils trop fréquents d'une part, mais à conserver une réactivité acceptable.

Dans le cadre de MusicSpace, plusieurs « mélodies » sont à jouer simultanément : le problème est donc légèrement plus compliqué et se pose alors de savoir si l'on conserve une seule tâche qui s'occupe de l'ensemble des mélodies, ou bien si au contraire un processus est construit par mélodie. Nous avons choisi de conserver une seule tâche qui s'occupe de l'ensemble des mélodies à jouer : à chaque réveil, la tâche parcourt les différentes mélodie et transmet au système les événements de chacune d'entre elle qui occurrent dans la fenêtre d'activation.

Notre projet ne gère que des mélodies « statiques », c'est-à-dire dont l'ensemble des notes est déterminé au chargement du fichier musical et reste inchangé. Dans le cas de mélodies plus « dynamiques », c'est-à-dire dont certaines notes peuvent être ajoutées ou ôtées en cours d'exécution, une réflexion plus poussée doit être faite sur l'architecture du séquenceur. Ces cas permettaient par exemple d'ajouter un « écho » en fonction de la position des sources sonores par rapport à l'avatar, ou encore de filtrer ces mélodies pour en enlever des éléments : par exemple les appogiatures et ornements d'une mélodie pourraient être supprimées à mesure que la source s'éloigne. L'éloignement ne résulterait donc plus seulement en un procédé de mixage mais aurait également des répercussions au niveau de l'arrangement musical en procédant à une simplification ou appauvrissement de la mélodie. Il n'a malheureusement pas été possible, dans cette recherche, et par manque de temps d'explorer ce type de possibilités musicales.

Apprentissage des fonctions MIDI

L'apprentissage des fonctions MIDI a pour but de produire au sein de notre application un moyen simple de contrôler n'importe quel paramètre d'un appareil externe disposant d'une connexion MIDI. Nous avons pour cela imaginé des objets graphiques capables de transmettre des valeurs de contrôle à partir de leurs paramètres géométriques de position, exprimés en coordonnées polaire ou cartésienne : ces objets formatent ces valeurs de contrôle au sein de messages MIDI intelligibles pour le module externe connecté.

Le problème du contrôle MIDI est que les messages à transmettre sont souvent fastidieux à décrire. Dans les cas simples, le pilotage d'un contrôleur MIDI par exemple, le message correspondant est une suite standard de trois octets décrivant le type de message et le numéro de canal MIDI utilisé pour le premier, le numéro du contrôleur concerné pour le deuxième et la valeur de paramètre à transmettre pour le dernier. Malheureusement ces messages standards sont insuffisants : ils ne permettent de contrôler qu'un nombre trop faible de paramètres, comparé par exemple au nombre de ceux d'une console de mixage. Il est alors nécessaire d'avoir recours aux messages dit « exclusifs », non standards, différents pour chaque machine, et beaucoup plus difficiles à extraire de la « table d'implémentation MIDI », même pour une personne avisée.

Le synopsis d'utilisation de nos objets MIDI « universels » est donc le suivant : l'utilisateur commence par construire un tel objet et le positionne sur l'écran. Il désigne ensuite quelle variable de l'objet va modifier la valeur du paramètre qu'il souhaite contrôler et effectue un choix entre la distance, l'azimut ou l'écart suivant les axes X ou Y en valeurs relatives ou absolue de l'objet par rapport à la position de l'avatar. Le système passe alors en mode « apprentissage » et enregistre tous les messages MIDI qui lui sont transmis et en particulier ceux produits lorsque l'utilisateur fait varier le paramètre qu'il souhaite contrôler sur le module externe, suivant tout son ambitus de valeurs : il est important pendant cette phase d'enregistrement que tous les autres appareils MIDI présents restent silencieux sur la boucle MIDI. Notre système analyse alors la séquence de messages pour séparer la partie variable de la partie constante de ces messages, celle qui détermine l'action de modification et l'adresse du paramètre, de celle qui détermine la valeur du paramètre lui même.

Ce principe repose sur la propriété des systèmes MIDI à utiliser le même message pour décrire une l'action et son résultat. Typiquement une console de mixage automatisée utilisera le même message MIDI pour signaler que la valeur d'un de ses potentiomètres a été modifiée par un utilisateur ainsi que pour mettre à jour la valeur d'un de ses potentiomètres. S'il est possible de s'appuyer sur cette propriété, c'est qu'elle est nécessaire à la compatibilité des systèmes aux logiciels de séquencement : c'est évident dans le cas d'un clavier-synthétiseur MIDI par exemple, pour lequel le message sortant du synthétiseur et correspondant à « la touche numéro 60 a été enfoncée » est le même que le message entrant dans le synthétiseur et ordonnant : « jouer la note DO ». C'est également le cas pour les consoles de mixages numériques pour lesquelles le logiciel de séquencement peut faire facilement office d'automation.

Les messages récoltés sont donc analysés pour déterminer quelle en est la partie variable. Nous présupposons qu'un seul paramètre est modifié à la fois et donc que chacun des octets

« variables » du message contribuent à la définition de la valeur de ce paramètre. Nous présupposons également (ce qui est toujours le cas) que ce nombre est écrit sur plusieurs « digit », dans une base quelconque. En s'appuyant sur le fait que le paramètre à contrôler a été modifié continûment, notre système examine donc ces variations pour déterminer quelles est la fonction de chaque octet variable, quelle est l'ambitus de variations de ce paramètres et en particulier si ce paramètre peut prendre des valeurs négatives ou non (à l'examen du bit de poids fort).

Un exemple typique de résultat obtenu est celui présenté sur la Figure 94 en page 130, où un objet MIDI universel contrôle un filtre de la console de mixage en amplitude à partir de sa distance à l'avatar, et en amplitude à partir de son azimut. Le message transmis est, par exemple pour l'amplitude, le suivant :

240 67 16 61 0 6 40 D2 D1 247

où D2 et D1 sont les deux octets variables du message, exprimant la valeur de contrôle à partir d'un comptage en base 16, et sur une plage de valeurs variant de -20 à 36 et correspondant à la plage explorée par l'utilisateur lors de l'enregistrement des messages d'apprentissage. Ce type d'objet s'avère très utile et permet de mettre en œuvre des situations de contrôle de manière aisée et rapide tel que cela a été présenté dans une expérience de contrôle de MusicSpace par un dataglove et le convertisseur analogique vers MIDI atomic-pro de l'IRCAM (voir [Fléty, 2002], [Fléty & Serra, 1998] et [AtoMIC pro]) lors de la conférence ICMC 1999 (International Computer Music Conference) à Pékin.

OpenSpace.

OpenSpace [Delerue & Agon, 1999] est une expérience menée en collaboration avec Carlos Agon de l'équipe Représentations Musicales de l'IRCAM afin d'étudier dans quelle mesure la combinaison de notre prototype MusicSpace, particulièrement adapté au contrôle en temps réel, avec un environnement visuel de programmation musicale OpenMusic plus apte à définir conceptuellement la scène à contrôller, permet d'élargir les possibilités d'utilisation de chacune des deux applications.

Un ensemble de classes correspondant aux objets que l'on souhaite instancier dans MusicSpace ont été définies dans OpenMusic. Ces classes, représentées en Figure 104, héritent toutes de la classe « eventmidi » de sorte que leur évaluation produise le code d'un message MIDI que MusicSpace saura décrypter.

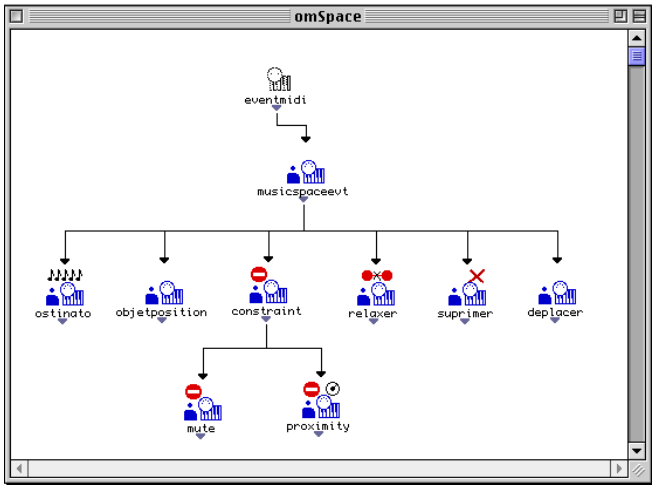


Figure 104 : classes MusicSpace construites dans OpenMusic

Par exemple, l'objet « ostinato » possède 5 paramètres : un canal MIDI, une position en abscisse, une position en ordonnée, un paramètre de hauteur MIDI et une période exprimée en millisecondes. Lorsque cet objet est évalué, un message MIDI est généré, correspondant à la construction d'un objet équivalant dans MusicSpace, aux positions spécifiées, et jouant périodiquement la note précisée, au tempo défini par le message, et avec une intensité inversement proportionnelle à la distance séparant cet objet de l'avatar.

Le « patch » présenté en Figure 105 permet de calculer un ensemble de messages de créations d'objet « ostinato », positionnés aléatoirement dans un rectangle (paramètres ([x0,y0],[x1,y1]) définissant les coins supérieur gauche et inférieur droit du rectangle, dont les hauteurs sont les multiples entiers (traduit et arrondis en valeurs MIDI) d'une hauteur de base. L'idée est donc de générer une scène sonore dans laquelle l'espace est découpé en régions contenant des objets sonores constituant un spectre sonore. L'exploration de cet espace génère donc des variations

dans la couleur spectrale du résultat sonore.

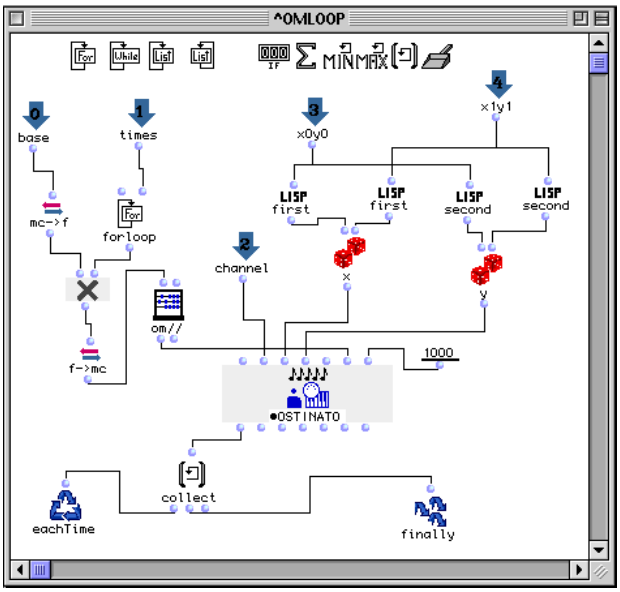


Figure 105 : construction de patchs dont l'évaluation produit des objets dans MusicSpace

L'éditeur de maquette d'OpenMusic permet de mettre « en temps » les générations d'objets, et donc de construire un scénario. La Figure 106 donne un exemple d'une telle définition temporelle : pendant les 5 premières secondes les objets Ostinato sont crées un à un. Puis, pendant les cinq secondes suivantes un ensemble de contraintes est ajouté aux objets ostinato. La scène peut alors être explorée pendant quelques temps, jusqu'à la seconde 16 où les objets commencent à être progressivement détruits. Bien entendu, cette scène peut ensuite être reconstituée directement dans MusicSpace, sans avoir recours à OpenMusic pour peu que l'ensemble des messages générant la construction, puis la suppression des objets aie été sauvegardé dans un fichier MIDI.

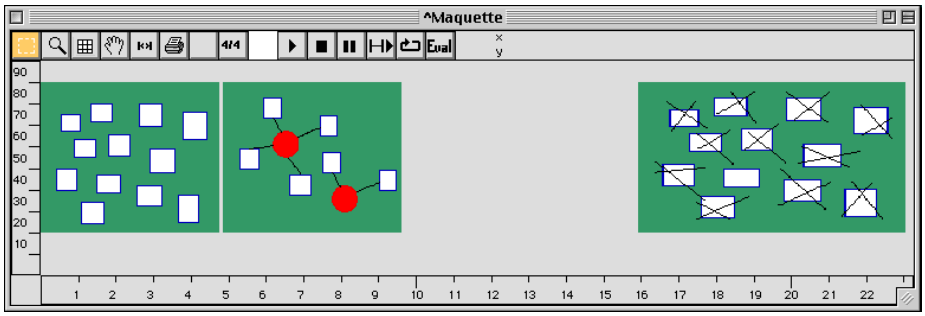


Figure 106 : définition temporelle des patchers grâce à l'éditeur de maquettes

Pour finir, la Figure 107 donne un aperçu de la scène construite dans MusicSpace : les objets ostinato y sont représentés par des carrés de couleurs. Pour chaque groupe d'ostinato, des contraintes de « groupe de mute » et contraintes de « proximité » ont été ajoutées de telle sorte que l'espace que l'utilisateur peut explorer soit vraiment divisé en sous parties correspondant à chacun des ensembles harmoniques.

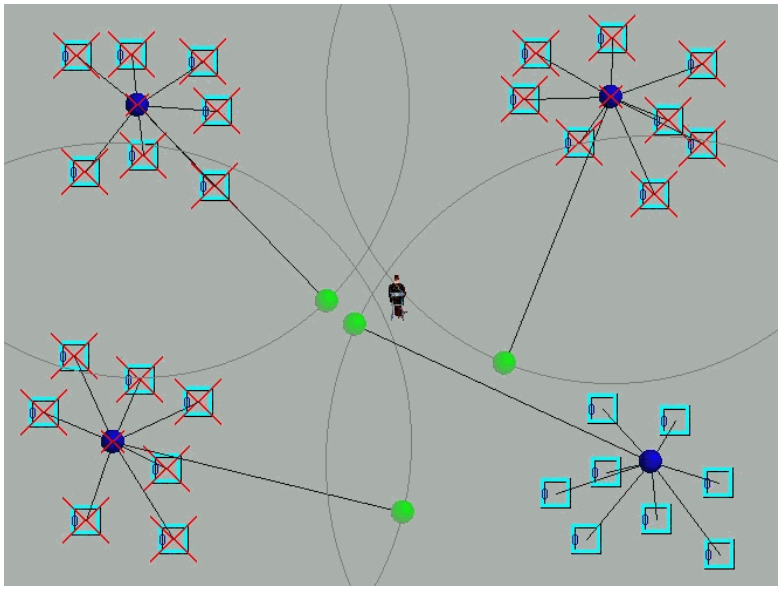


Figure 107 : aperçu des objets construits dans MusicSpace

Cette expérience met en évidence plusieurs points. Tout d'abord elle montre l'intérêt de l'aspect incrémental de l'algorithme de contraintes, selon lequel il est possible d'ajouter ou supprimer dynamiquement des contraintes au système. Le « comportement » des objets ou la manière dont il réagissent les uns par rapport aux autres peut donc être décidé « à la volée », à partir de paramètres de composition de la scène.

Par ailleurs, cette expérience montre l'intérêt que notre système peut apporter à d'autres domaines, comme tout particulièrement celui de la synthèse sonore où il est bien souvent question de contrôler un modèle de synthèse à partir d'un jeu de paramètres relativement complexe. Comme nous avons essayé de le montrer, le domaine précis de la synthèse additive se prête remarquablement bien au jeu de l'exploration et à l'idée d'établir des relations entre les paramètres originaux, tous de même nature, et en nombre important.

Bibliographie

Musique et références en ligne :

[Rodgers & Hart, 1938] Richard Rodgers (musique) et Lorenz Hart (lyriques).
« Falling in love with love »
De la comédie musicale « The Boys From Syracuse », 1938.
Publié dans le « Real Book ».

[Wes, 1996] Welenga (Universal Consciousness)
Album 1996 CD Europe (France) Epic/Sony SAN 485146-2

[EAGLE] Téléchargement et documentation en ligne :
http://developer.creative.com/LEFTMENU/DC_LM_Downloads.HTML

[VDesk] JLCooper Electronics
http://www.soundtech.co.uk/jlcooper/vdesk.htm

[DirectX]
http://www.microsoft.com/windows/directx/default.asp

[General Midi, 1991] Midi Manufacturer Association
http://www.midi.org/

[AtoMIC pro] Analogic TO MIdi Converter
http://www.ircam.fr/produits/technologies/atomic/Atomic-e.html

[OpenAl] Open Audio Library
http://www.openal.org

[MMA, 1998] MIDI Manufacturers Association
3D Audio Rendering and Evaluation Guideline, revision 1.0
Document disponible en ligne sur le site internet de [IA-Sig]

[MMA, 1999] MIDI Manufacturers Association
3D Audio Rendering and Evaluation Guideline, Level 2.0, révision 1.0a
Document disponible en ligne sur le site internet de [IA-Sig]

[IA-Sig] Interactive Audio Special Interest Group.
http://www.iasig.org/

Articles

[Agon & Al, 1998] Carlos Agon, Gérard Assayag, Olivier Delerue, Camilo Rueda
« Objects, Time and Constraints in OpenMusic. »
Actes de l'International Computer Music Conference ICMC98, Ann Arbor 1998.

[Allen, 1983] J. Allen
Maintaining Knowledge about Temporal Intervals
in CACM, vol 26, no. 11, pp 832 – 843, Novembre 1983.

[Assayag & Al, 1999] Gérard Assayag, Camilo Rueda , Mikael Laurson, Carlos Agon, Olivier Delerue.
« Computer Assisted Composition at Ircam : PatchWork & OpenMusic. »
in Computer Music Journal 23:3, Fall 1999, MIT Press, 1999.

[Assayag & Al, 1999b] Gérard Assayag, Shlomo Dubnov, Olivier Delerue
« Guessing the Composer's Mind: Applying Universal Prediction to Musical Style »
Proceedings of the 1999 International Computer Music ICMC 99, Beijing 1999, pp. 496-499.

[Badros & Borning, 2001] Greg J. Badros, Alan Borning, and Peter J. Stuckey,
« The Cassowary Linear Arithmetic Constraint Solving Algorithm »
in ACM Transactions on Computer Human Interaction, Vol. 8 No. 4, December 2001, pp. 267-306.

[Badros & Borning, 1998] Greg J. Badros, Alan Borning
« The Cassowary Linear Arithmetic Constraint Solving Algorithm : Interface and Implementation »
Technical Report UW-CSE-98-06-04, June 1998.

[Ballesta, 1998] Philippe Ballesta
« Contraintes et objets, clefs de voute d'un outil d'aide à la composition ? »
in *Recherches et applications en informatique musicale*, éditions Hermes, pp. 75 – 92, Paris, 1998.

[Begault, 2000] Duran R. Begault
« 3-D Sound for Virtual Reality and Multimedia »
NASA/TM-2000-000000, Ames Research Center, Moffett Field, California 94035, 2000

[Beurivé, 2000] Anthony Beurivé
« Un logiciel de composition musicale combinant un modèle spectral, des structures hiérarchiques et des contraintes. »
Actes des Journées d'informatique Musicale, JIM 2000, Bordeaux, 2000, pp 21-30.

[Blauert, 1999] Jens Blauert
« Spatial Hearing – The Psychophysics of Human Sound Localization »
MIT Press, Revised Edition, Second printing, 1999, ISBN 0-262-02413-6

[Bliek & Al, 1998] Christian Bliek, Bertrand Neveu, Gilles Trombettoni
« Using Graph Decomposition for Solving Continuous CSPs »
in Principles and Practice of Constraint Programming, CP'98, Springer LNCS 1520, pp.102 – 116, 1998.

[Bonnet et Rueda, 1998] Antoine Bonnet, Camilio Rueda.
« Un langage visuel basé sur les contraintes pour la composition musicale. »
in *Recherches et applications en informatique musicale*, éditions Hermes, pp. 75 – 92, Paris, 1998.

[Borning, 1981] A. Borning,
« The Programming Language Aspects of ThingLab, a Constraint-oriented Simulation Laboratory »
ACM Transactions on Programming Languages and Systems, vol. 3(4), pp. 353-387, 1981.

[Borning & Al., 1992] Alan Borning, Bjorn Freeman-Benson, Molly Wilson
« Constraint Hierachies »
in LISP AND SYMBOLIC COMPUTATION: An International Journal, 5, pp. 223 – 270, 1992

[Borning & Freeman-Benson, 1995] A. Borning and B. Freeman-Benson,
« The OTI Constraint Solver: A Constraint Library for Constructing Interactive Graphical User Interface »
Actes de Principles and Practice of Constraint Programming, CP'95, Cassis, France, 1995.

[Borning, & Al., 1996] A. Borning, R. Anderson, and B. Freemann-Benson,
« Indigo : A Local Propagation Algorithm for Inequality Constraints »
Actes de ACM Symposium on User Interface Software and Technology, 1996.

[Borning & Al, 1997] Alan Borning, Richard Lin, Kim Marriott
« Constraint for the web »
Proceedings of the fifth ACM International Multimedia Conference, 1997, Seattle, USA.

[Borning et Al., 1997b] Alan Borning, Kim Marriott, Peter Stuckey, Yi Xiao
« Solving Linear Arithmetic Constraints for User Interface Application. »
Proceedings of the 1997 ACM Symposium on User Interface and Technology, 1997, pp. 87-96.

[Borning & Freeman-Benson, 1998] Alan Borning, Bjorn N. Freeman-Benson
Ultraviolet: A Constraint Satisfaction Algorithm for Interactive Graphics
In CONSTRAINTS: An International Journal, Special Issue on Constraints, Graphics, and Visualization, Vol. 3 No. 1, April 1998, pages 9-32.

[Burgess, 1992] David A. Burgess
« Techniques for Low Cost Spatial Audio »,
ACM Fifth Annual Symposium on User Interface Software and Technology (UIST '92), pp53-59, Monterey, November 1992.

[Cabaud & Pottier, 2002] Benjamin Cabaud, Laurent Pottier
« Le contrôle de la spatialisation multi-sources – Nouvelles fonctionnalités dans Holophon version 2.2 »
Actes des Journées d'Informatique Musicale, 9ème Edition, Marseille, pp 269 – 271, 2002.

[Cerveau, 1999] Laurent Cerveau
« Couplage temps réel d'outils d'acoustique prévisionnelle et de dispositif d'auralisation »
Thèse de doctorat, université Paris 6, 1999.

[Chemillier, 1999] Marc Chemillier.
« Ethnomusicologie, ethnomathématique. Les logiques sous-jacentes aux pratiques artistiques transmises oralement »
Forum Diderot, Société Mathématique Européenne, 1999

[Chemillier & Truchet, 2001] Marc Chemillier, Charlotte Truchet
« Two Musical CSPs »
Actes de Musical Constraints Workshop, CP 01, Cyprus 2001

[Chemillier & Al., 2002] Marc Chemillier, Charlotte Truchet Louis-Martin Rousseau
« Analyse musicale et contraintes »
Actes des Journées d'Informatique Musicale, JIM 2002, Marseille, 2002.

[Chowning, 1971] John Chowning
« The simulation of moving sound sources »
JAES, Journal of the Audio Engineering Society, Volume 19, N°1, pp. 2 – 6, 1971.

[Codognet, 2000] Philippe Codognet
« Adaptative search, preliminary results »
Book of the 4th ERCIM/CompulogNet Workshop, 2000.

[Daniel, 2000] Jérome Daniel.
« Représentation de champs acoustiques, application à la transmission et à la reproduction de scènes sonores complexes dans un contexte multimédia »
Thèse de doctorat, université Paris 6, Juillet 2000.

[Debussy, 1903] Claude Debussy
« Considérations sur la musique en plein air » dans *Monsieur Croche et autres écrits*, Paris, Gallimard, 1987, pp. 74 – 76.

[Deflache, 2001] Nicolas Deflache.
« Réalisation d'un mixage de pop-music interactif »
Mémoire de fin d'études – Formation Supérieure aux Métiers du son – CNSM Paris.

[Delerue, 1997] Olivier Delerue
« Etude et réalisation d'opérateurs musicaux pour un environnement de composition assistée par ordinateur »
DEA ATIAM, Ircam - Université Paris VI, 1997. Sous la direction de Gérard Assayag.

[Delerue & Lorrain, 1998] Olivier Delerue, Denis Lorrain
« Le Spatialisateur SONVS »
in *Le son et l'espace*, collection ALÉAS – GRAME, ISBN 2-908016-96-6, pp 157 – 162

[Delerue & Pachet, 1998] Olivier Delerue, François Pachet
« Constraint Propagation for real time spatialization »
ECAI 98 Workshop on Constraints for Artistic Applications, Brighton, 1998.

[Delerue & Pachet, 1998b] Olivier Delerue, François Pachet
« MidiSpace, un spatialisateur midi expérimental »
Actes des Journées d'informatique musicale JIM 98, Agelonde 1998

[Delerue & Al., 1998] Olivier Delerue, Gérard Assayag, Carlos Agon
« Etude et réalisation d'opérateurs rythmiques dans OpenMusic, un environnement de programmation appliqué à la composition musicale »
Actes des Journées d'informatique musicale JIM 98, Agelonde 1998

[Delerue & Agon, 1999] Olivier Delerue, Carlos Agon
« OpenMusic + MusicSpace = OpenSpace »
Actes des Journées d'informatique musicale, JIM 99, Issy-les-moulineaux, pp 89 – 96, 1999 .

[Delerue & Warusfel, 2002] Olivier Delerue, Olivier Warusfel.
« Authoring of virtual sound scenes in the context of the LISTEN project »
Actes de la 16ème conference de l'Audio Engeeniring Society, AES16, Espoo Finland, 2002.

[Ebcioglu, 1988] Kemal Ebcioglu
« An Expert System for Harmonizing Four-part Chorales,
Computer Music Journal, Volume 12, Number 3, pp. 43 - 51, MIT Press, 1988.

[Fléty & Serra, 1998] Emmanuel Fléty, Marie-Hélène Serra
« Recent use of gestural sensors for Ircam musical creations »
Actes des Journées d'Informatique Musicale, JIM98, La Londe-les-Maures, 1998, pp D3-1 – D3-4.

[Fléty, 2002] Emmanuel Fléty
« AtoMIC Pro : a Multiple Sensor Acquisition Device »
Actes de la conférence NIME-02 (New Interfaces for Musical Expression) Dublin, Ireland, pp 96 – 101, 2002.

[Freeman-Benson & Maloney, 1989] Bjorn Freeman-Benson, John Maloney
« The DeltaBlue algorithm : An Incremental Constraint Hierarchy Solver »
Proceedings of the Eighth Annual IEEE Phoenix Conference on Computers and Communications, pp 561 – 568, Scottsdale, Arizona, Mars 1989.

[Freeman-Benson & Al., 1990] Bjorn Freeman-Benson, John Maloney, Alan Borning
« An Incremental Constraint Solver »
Communications of the ACM, 33(1):54-63, Janvier 1990.

[Freeman-Benson, 1993] Bjorn Freeman-Benson.
« Converting an existing user interface to use constraints »
In Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology, pages 207–215, Atlanta, Georgia, November 1993.

[Galliari, 1994] Alain Galliari
« Petit historique de l'enregistrement »
Revue Résonance nº 6, mars 1994, Copyright © Ircam - Centre Georges-Pompidou 1994

[Gerzon, 1972] Michael A. Gerzon
« Periphony: Width-Height Sound Reproduction »
Journal of the Audio Engineering Society, 21(1), pp. 2-10, 1972

[Gosling, 1983] James Gosling
« Algebraic Constraints »
Thèse de doctorat, Department of Computer Science, Carnegie-Mellon University, May 1983.

[Helm & Al., 1995] Richard Helm, Tien Huynh, Kim Marriott, John M. Vlissides
« An Object-Oriented Architecture for Constraint-Based Graphical Editing »
in Object-Oriented Programming for graphics, pp 217 – 238, Springer Verlag, 1995.

[Hanappe, 2000] Peter Hanappe
« Design and implementation of an integrated environment for music composition and synthesis »
Thèse de doctorat, Université Paris 6, 2000

[Harvey & Al., 1997] Warwick Harvey, Peter Stuckey, Alan Borning
Compiling constraint solving using projection
in Proceedings of the 1997 Conference on Principles and Practice of Constraint Programming (CP97), Octobre 1997.

[Hower & Winfried, 1996] Walter Hower, Winfried H. Graf
« A bibliographical survey of constraint-based approaches to CAD, graphics, layout, visualization, and related topics »
In Knowledge-Based Systems 9, pp.449-464, 1996

[Jaffar & Al, 1992] Joxan Jaffar, Spiro Michaylov, Peter Stuckey, Roland Yap
« The CLP(R) language and system »
in ACM Transactions on Programming Languages and Systems, 14(3):339-395,Juillet 1992

[Jot, 1999] Jean-Marc Jot
« Real-time spatial processing for music, multimedia and interactive human-computer interfaces »
in Multimedia Systems 7 pp55-69, Springer-Verlag 1999.

[Jot & Al, 1999] Jean-Marc Jot, Véronique Larcher, Jean-Marie Pernaux
« A comparative Study of 3-D audio encoding and rendering techniques »
Actes de la conférence AES 16th International Conference on Spatial Sound Reproduction, Rovaniemi, pp 281 – 300, 1999.

[Jourdan & Al., 1998] Muriel Jourdan, Nabil Layaïda, Cécile Roisin, Loay Sabry-Ismaïl, Laurent Tardif.
« Madeus, an Authoring Environment for Interactive Multimedia Documents »
Actes de la 6ème conference ACM international conference on Multimedia, MULTIMEDIA '98. September12-16,1998, Bristol, England, pp267-272

[Jourdan & Al., 1998b] Muriel Jourdan, Cécile Roisin, Laurent Tardif
« Constraints Techniques for Authoring Multimedia Documents »
ECAI 98 Workshop on Constraints for artistic applications, Brighton (UK), Août 1998.

[Jullien, 1995] J.P. Jullien
« Structured model for the representation and the control of room acoustical quality »
in Proceedings of the 15th International Conference on Acoustics, 1995, pp517-520.

[Koenen, 1999] Rob Koenen.
« MPEG-4 Multimedia for our time »
Revue IEEE Spectrum, Vol 36 N°2, février 1999.

[Larcher, 2001] Véronique Larcher
« Techniques de spatialisation du son pour la réalité virtuelle »
Thèse de doctorat, Université Paris VI, 2001.

[Laurson, 1993] Michael Laurson
« PWConstraints »
In Haus G. and Pighi I. (eds), X Colloquio di Informatica Musicale, 332-335, Associazione di Informatica Musicale Italiana, . Milano 1993.

[Malham & Myatt, 1995] David G. Malham, Anthony Myatt
« 3-D Sound Spatialization using Ambisonic Techniques »
in Computer Music Journal, 19:4, pp. 58 – 70, 1995.

[Maloney, 1991] John Maloney
« Using Constraints for User Interface Construction »
Thèse de doctorat, Department of Computer Science and Engineering, University of Washington, August 1991. Published as Department of Computer Science and Engineering Technical Report 91-08-12.

[Marriott & Al., 1998] Kim Marriott, Sitt Sen Chok, Alan Finlay
« A tableau based constraint solving toolkit for interactive graphical applications »
In International Conference on Principles and Practice of Constraint Programming (CP98), pp. 340 – 354, 1998.

[Moore, 1990] F. R. Moore
« Elements of Computer Music»
Englewood Cliffs, New Jersey 07362: Prentice Hall, 1990.

[Myers & Al., 1990] Brad A. Myers, Dario Giuse, Roger B. Dannenberg, Brad Vander Zanden, David Kosbie, Ed Pervin, Andrew Mickish, Philippe Marchal
« Garnet; Comprehensive Support for Graphical, Highly-Interactive User Interfaces »
IEEE Computer. Vol. 23, No. 11. November, 1990. pp. 71-85.

[Nicol, 1999] Rozenn Nicol
« Etude de la restitution du son spatialisée en zone étendue : Application à la téléprésence »
Thèse de doctorat, Université du Maine, 1999.

[Nouno & Agon, 2002] Gilbert Nouno, Carlos Agon.
« Contrôle de la spatialisation comme paramètre musical »
Actes des Journées d'Informatique Musicale, 9ème édition, Marseille, pp 115 – 118, 2002

[Orlarey & Lequay, 1989] Yann Orlarey, Hervé Lequay.
« MidiShare: a real time multi-tasks software module for Midi applications »
Actes de l'International Computer Music Conférence, ICMC 1989, San Francisco, 1989.

[Pachet & Delerue, 1998] François Pachet, Olivier Delerue
« Annotations for Real Time Music Spatialization »
International Workshop on Knowledge Representation for Interactive Multimedia Systems, KRIMS-II workshop, Trento, Italy, 1998.

[Pachet & Delerue, 1998b] François Pachet, Olivier Delerue
« Constraint Based Spatialization »
Actes de la conférence DAFX98, Barcelone, Novembre 1998, pp 71 – 75.

[Pachet & Delerue, 1998c] François Pachet, Olivier Delerue
« MidiSpace: a temporal constraint-based music spatializer »
Actes de la 6ème conference ACM international conference on Multimedia, MULTIMEDIA '98, September12-16,1998, Bristol, England, pp351-359

[Pachet & Delerue, 1998d] François Pachet, Olivier Delerue
« A Mixed 2D/3D Interface for Music Spatialization »
First International Conference on virtual Worlds (VW98), Springer-Verlag, Lecture Notes in Computer Science, n. 1434, pp. 298-307.

[Pachet & Delerue, 1999] François Pachet, Olivier Delerue
« MusicSpace : a Constraint-Based Control System for Music Spatialization »
Proceedings of the 1999 International Computer Music ICMC 99, Beijing 1999, pp. 272-275.

[Pachet & Roy, 2001] François Pachet, Pierre Roy
Musical Harmonization with Constraints : A Survey
in Constraints Journal, Kluwer Publisher, 6(1), pp. 7 – 19, 2001

[Pachet & Al., 2000] François Pachet, Olivier Delerue, Peter Hanappe,
« Dynamic Audio Mixing »
actes de l'International Computer Music Conference, ICMC2000, Berlin 2000.

[Pachet & Al., 2000b] François Pachet, Olivier Delerue, Peter Hanappe.
« MusicSpace goes audio »
Sound in Space Conference, CREATE, Santa Barbara, March 2000.

[Périaux, 2000] Bergame Périaux
« Cohérence de l'image sonore en mixage variétés 5.1 »
Mémoire de fin d'études – Formation Supérieure aux Métiers du son – CNSM Paris.

[Pernaux & Al, 1998] Jean-Marie Pernaux, Patrick Boussard, Jean-Marc Jot
« Virtual Sound Source Positioning and Mixing in 5.1. Implementation on the Real-Time System Genesis »
Actes de la conférence DAFX98, Barcelone, Novembre 1998, pp 76 – 80.

[Pottier, 2000] Laurent Pottier
« Holophon : projet de spatialization multi-sources pour une diffusion multi-haut-parleurs »
Actes des Journées d'Informatique Musicale, JIM 2000, Bordeaux, pp 96 – 102, 2000.

[Pottier, 1998] Laurent Pottier
« Dynamical Spatialisation of sound. HOLOPHON : a graphical and algorithmical editor for ∑1 »
Actes de la conférence DAFX98, Barcelone, Novembre 1998, pp 81 – 84.

[Puckette, 1988] Miller Puckette
« The Patcher »
Proceedings, ICMC International Computer Music Conference, San Francisco, pp. 420-429, 1988

[Puget, 1994] J.-F. Puget,
« Programmation sous contraintes en C++»
Actes de « Langages et modèles à objets » LMO'94, Grenoble (France), 1994

[Pulkki, 2001] Ville Pulkki
« Spatial Sound Generation and Perception by Amplitude Panning Techniques »
Thèse de doctorat, Helsinki University of Technology, ISBN 951-22-5531-6
Référence en ligne : http://www.acoustics.hut.fi/~ville/

[Pulkki & Al, 1999] Ville Pulkki, M. Karjalainen, and J. Huopaniemi.
« Analyzing virtual sound source attributes using a binaural auditory model. »
Journal of the Audio Engineering Society, 47(4) pp. 203-217 April 1999 .

[Pulkki, 1998] Ville Pulkki.
« Creating generic sound scapes in multichannel loudspeaker systems using vector base amplitude panning in csound synthesis software. »
Journal Organised Sound, 3(2) pp. 129-134, 1998.

[Pulkki, 1997] Ville Pulkki.
« Virtual sound source positioning using vector base amplitude panning »
Journal of the Audio Engineering Society, 45(6) pp. 456-466, June 1997.

[Risset & Al, 2002] Jean-Claude Risset, Daniel Arfin, Antonio de Sousa Dias, Denis Lorrain, Laurent Pottier
« De Inharmonique à Resonant Sound Spaces : temps réel et mise en espace »
Actes des Journées d'Informatique Musicale, 9ème Edition, Marseille, 2002, pp 83 – 88.

[Roy, 1998] Pierre Roy.
« Satisfaction de contraintes et programmation par objets »
Thèse de doctorat, Université Paris VI, 1998

[Sabri-Ismail & Guetari, 1998] Loay Sabri-Ismail et Ramzi Guetari
« Le Modèle Objet Madeus »
in L'objet, N°2, vol 4, Editions Hermes, pp 155 – 171, 1998.

[Sannella, 1993] Michael Sannella
« The SkyBlue Constraint Solver and Its Applications. »
Proceedings of the 1993 workshop on Principles and Practice of Constraint Programming, MIT Press, 1994.

[Sannella & Al., 1993b] Michael Sannella, John Maloney, Bjorn Freeman-Benson, Alan Borning
« Multi-way versus One-way Constraints in User Interfaces : Experience with the DeltaBlue algorithm »
In Software – Practice and Experience, Vol 23 No. 5 (may 1993), pages 529-566.

[Sanella, 1994] Michael Sannella
« Constraint satisfaction and debugging for interactive user interfaces »
Thèse de doctorat, Department of Computer Science and Engineering, University of Washington, Seattle, 1994. (Disponible sous la forme du Technical Report 94-09-10)

[Solomos, 1995] Makis Solomos
« Musique, Son Espace »
Actes du colloque Rencontres Pluridisciplinaires Informatique et Musique, « Le Son et L'espace », Lyon, 1995, pp.69 – 76.

[Sutherland, 1963] I. Sutherland:
« Sketchpad: a man-machine graphical communication system »
Actes de l'IFIP Spring Joint Conference, 1963

[Todoroff & Traube, 1996] Todor Todoroff, Caroline Traube
« Interfaces graphiques NeXTSTEP pour la commande d¹instruments virtuels d¹aide à l¹interprétation de musique électroacoustique. »
Actes des Troisièmes Journées d¹Informatique Musicale (JIM96), Caen, France, pp. 98 – 102, 1996.

[Todoroff & Al, 1997] ." Todor Todoroff, Caroline Traube, Jean-Marc Ledent
« NeXTSTEP Graphical Interfaces to Control Sound Processing and Spatialization instruments. »
Actes de l' International Computer Music Conference (ICMC), Thessaloniki, Grèce, pp. 325 – 328, 1997.

[Trombettoni & Neveu, 1997] Gilles Trombettoni, Bertrand Neveu
Computational Complexity of Multi-Way, Dataflow Constraint Problems.
Proceedings of the fifteenth International Joint Conference on Artificial Intelligence IJCAI 97, Nagoya, Aichi, Japan, 1997, pages 358-363.

[Trombettoni, 1997b] Gilles Trombettoni
« Algorithmes de maintien de solution par propagation locale pour les systèmes de contraintes.
Thèse de doctorat, Université de Nice Sophia-Antipolis, juin 1997.

[Truchet & Al., 2001] Charlotte Truchet, Gérard Assayag, Philippe Codognet
« Visual and Adaptive Constraint Programming in Music »
Actes de l'Internation Computer Muisc Confernece, ICMC 01, La Havana, 2001

[Tsingos, 1998] Nicolas Tsingos
« Simulation de champs sonores de haute qualité pour des applications graphiques interactives»
Thèse de doctorat de l'université Joseph Fourier, Grenoble, 1998.

[Viara & Puckette, 1990] Eric Viara, Miller Puckette
« A real-time operating system for computer music »
Actes de l'International Computer Music Conference, ICMC 90, Glasgow, 1990

[Vander Zanden, 1996] Bradley T. Vander Zanden
« An Incremental Algorithm for Satisfying Hierarchies of Multi-way, Dataflow Constraints »
in ACM Transactions on Programming Languages and Systems, 18(1), pp. 30-72, 1996

[VanderZander & Myers, 1991] Brad Vander Zanden , Brad A. Myers
« The Lapidary graphical interface design tool »
Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology, March 1991

[DeVries & Boone, 1999] D. De Vries, Marius M. Boone
« Wave Field Synthesis and analysis using array technology »
Proc. 1999 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, New Paltz, New York, Oct 17-20, 1999.