# These

présentée au

**Conservatoire National des Arts et Métiers**

pour obtenir le titre de

**Docteur en Informatique**

par

**Alexandre TOPOL**

Interactions 3D pour les paysages informationnels : application aux bibliothèques numériques et à la spécification de comportements sous VRML.

Soutenance le 18 Décembre devant le jury composé de

| | |
|---|---|
| **M. Marc Nanard** | **Président** |
| **M. Christophe Chaillou** | **Rapporteur** |
| **M. Jean-Daniel Fekete** | **Rapporteur** |
| **M. Didier Arques** | **Examinateur** |
| **M. Pierre Cubaud** | **Examinateur & Encadrant** |
| **M. Stéphane Natkin** | **Directeur de thèse** |

*À Panou et Manou*

*À Marie*

# *Remerciements*

Je tiens à remercier tous les membres de mon jury :

M. Marc Nanard, Professeur au LIRMM, qui m'a fait l'honneur de bien vouloir présider ce jury de thèse.

M. Christophe Chaillou, Professeur au LIFL et M. Jean-Daniel Fekete, Chargé de recherche à l'INRIA, qui ont accepté d'être rapporteurs et pour l'intérêt qu'ils ont porté à ce travail.

M. Didier Arquès, Professeur à l'Université de Marne-la-Vallée, d'avoir répondu favorablement à notre invitation dans ce jury.

M. Pierre Cubaud, Maître de Conférences au CNAM, pour m'avoir encadré, guidé et soutenu durant ces années de thèse.

M. Stéphane Natkin, Professeur au CNAM, pour avoir dirigé ma thèse.

Je remercie les membres de l'équipe Multimédia Interaction Homme-Machine et plus généralement ceux du Centre d'Etudes et de Recherches en Informatique du CNAM pour leur acceuil. J'en profite également pour remercier les personnes qui ont contribué, à différents niveaux, à l'élaboration de ma thèse : Jean-Marie Abisror, Yann Creach, Florent Schaeffer, Pascal Stokowski et Claire Thiria pour leur participation aux différents projets, FeuNeu pour sa relecture, Victor et Louis qui m'ont changé les idées et apporté un peu de fraîcheur.

Encore une fois, je remercie tout particulièrement Pierre Cubaud pour ses conseils et enseignements. Je souhaite que notre collaboration, entamée il y a six ans alors que j'arrivai au CNAM au niveau de la maîtrise, se prolonge au-delà de mon mémoire d'ingénieur et de cette thèse.

J'embrasse et je dédie également cette thèse à tous ceux qui m'ont poussé à suivre les cours du soir au Conservatoire National des Arts et Métiers alors que je n'avais plus le goût pour les études. En particulier, pour ma mère, mon père, Sabine, Catherine, Michel, Pierre et Rachel, merci de m'avoir aiguillé vers le CNAM et de m'avoir soutenu et supporté dans les périodes les plus difficiles.

# *Résumé*

Dans cette thèse, nous étudions les différentes alternatives aux interfaces 2D pour représenter et manipuler un grand nombre d'informations. Bien qu'un certain nombre de projets se soient attelés à la tâche d'améliorer la visualisation de données en proposant des solutions originales, le constat est néanmoins clair : les interfaces que nous utilisons quotidiennement sont toujours des déclinaisons de l'interface WIMP (*Windows Icon Menus Pointer*) inventée dans les années 70 au Xerox PARC.

Nous appliquons les techniques interactives 3D pour accéder aux bibliothèques numériques. Pour la recherche d'ouvrages, nous nous appuyons sur des heuristiques visuelles, reconnues comme étant des plus importantes, pour fournir une méthode de recherche non permise jusqu'alors. Pour le poste de lecture, nous avons travaillé sur la métaphore 3D du livre et sur les solutions techniques qui peuvent être employées. Nous avons également effectué une ébauche de bureau 3D pour faire évoluer la métaphore actuelle afin de fournir un environnement cohérent et unique capable de faire coexister les interfaces 2D et 3D.

VRML, le langage qui nous a servi dans ces différents projets, s'est avéré être insuffisant. Dès lors que les scènes sont fortement interactives, ce langage montre ses limites. Nous avons, de ce fait, proposé des extensions capables de gérer les interactions de nos applications. Afin de ne pas aller contre la nature généraliste du langage VRML, nous avons porté une attention particulière à ce que nos extensions soient aussi génériques que possibles, et utilisables dans le plus grand nombre d'applications. Cette solution s'est avérée intéressante pour limiter la part de scripts nécessaires pour gérer les interactions complexes et, ainsi, rendre à VRML, dans ce contexte, sa caractéristique de langage de description.

Motivés par cette même démarche, dans le cadre d'une étude sur la réalisation d'un projet artistique avec VRML, nous avons également proposé des extensions pour enrichir le modèle de spatialisation sonore. Ces extensions ont pour but de permettre la description de scènes audiovisuelles 3D et de gérer les aspects sonores en utilisant les techniques psychoacoustiques d'OpenAL. Pour ce faire, nous avons ajouté des propriétés sonores aux objets telles que leur taux de filtrage des sons et des attributs d'effets de salle associés à des groupes d'objets. Quant aux sources sonores déjà présentes dans la norme VRML, nous avons modifié leur description afin de les rendre plus complètes et intuitives.

## Mots-Clés :

Paysages informationnels, visualisation 3D, interaction 3D, bibliothèques numériques, VRML.

# *Abstract*

In this thesis, we study various solutions to represent and interact with numerous data. Even if there are an important number of projects which tried to propose original solutions for data visualization, we still don't use them currently. The interfaces that we use daily are always variations of WIMP interfaces ( *Windows Icon Menus To point* ) invented in the Seventies at the Xerox PARK.

We have studied the use of interactive 3D graphics to access to digital libraries. The first interface we have proposed is to search books. We have used the textures of books and a spatial organisation to enable visual heuristic which are known to be very efficient. For the second interface, the reading one, we worked on a 3D book metaphor and on technical solutions that can be employed. Beside studies on digital libraries, we also worked on creating a 3D workbench in order to provide a coherent and single environment shared by 2D and 3D applications.

VRML, the language we have used in these various projects, proved to be insufficient. When scenes are strongly interactive this language shows its limits. To enhance VRML, so it matches our needs, we proposed some simple extensions. To keep it a general purpose scene description language, we paid attention to make our extensions as generic as possible and usable in the greatest number of applications. This solution proved to be interesting to limit the number of scripts necessary to manage complex interactions. Thus, with highly interactive scenes, our extended VRML returns to a descriptive language.

To continue with enhancing the capabilities of the VRML language, we have also proposed some extensions to enrich its sound spatialization model. We started this study because we needed a 3D audio file format for an artistic project. The purpose of these extensions are to allow the description of 3D audio-visual scenes and to manage the sound aspects by using the psychoacoustic techniques of OpenAL. We added sound properties to the objects such as their filtering capabilities and some attributes to create some room effects for groups of objects. We have also changed the existing sound sources descriptions in the VRML standard in order to make them more complete and intuitive.

## Key Words :

Information Landscapes, 3D Visualisation, 3D Interaction, Digital Libraries, VRML.

# *Sommaire*


**1 Introduction** **1**

**2 Paysages informationnels** **5**

**3 Techniques et technologies** **61**

**4 Expérimentations** **129**

**5 Critiques et solutions proposées** **187**

**6 Conclusion et travaux futurs** **221**

**Publications** **231**

**Références** **233**

**Liste des acronymes** **251**

**Liste des figures** **253**

**Liste des tableaux** **257**

# *Table des matières détaillée*


**1 Introduction** **1**

**2 Paysages informationnels** **5**

2.1 Introduction du chapitre . . . . . 6

2.2 Les tâches couvertes par les paysages informationnels . . . . . 8

2.2.1 Trouver . . . . . 9

2.2.2 Appréhender le contenu . . . . . 10

2.2.3 Appréhender le contexte . . . . . 10

2.2.4 Organiser . . . . . 13

Cyberspace de Benedikt . . . . . 13

Partitionnement statistique et mesures de proximité . . . . . 13

Hyperstructures . . . . . 14

Approches centrées sur l'homme . . . . . 14

Comparaison et "applicabilité" . . . . . 15

2.2.5 Retrouver . . . . . 16

2.3 Les différents domaines d'application . . . . . 16

2.3.1 Les réseaux . . . . . 16

2.3.2 Les bases de données . . . . . 17

AMAZE . . . . . 17

WINONA . . . . . 18

2.3.3 Travail collaboratif . . . . . 19

Q-PIT . . . . . 20

BEAD . . . . . 20

VIBE et VR-VIBE . . . . . 22

LyberWorld . . . . . 24

Vineta . . . 27
VR-Mapper . . . 29
2.3.4 Les hypertextes . . . 30
Hyperspace . . . 31
WWW3D . . . 32
VizNet . . . 33
SemNet . . . 35
Web Forager . . . 36
Data Mountain . . . 37
Autres systèmes de visualisation 3D du WWW . . . 38
2.3.5 Les systèmes . . . 39
FSN . . . 40
VOGUE . . . 42
VisualLinda . . . 43
Graph Visualizer 3D . . . 44
Narcissus . . . 45
SHriMP Views . . . 46
2.3.6 Les ateliers de visualisation . . . 47
2.4 Autour de nos préoccupations . . . 48
2.4.1 Les bibliothèques numériques . . . 48
Activités et caractéristiques . . . 49
Les précurseurs . . . 50
Les recherches en cours . . . 52
Un autre Web . . . 52
Les informations hétéroclites . . . 53
La visualisation . . . 54
2.4.2 Les bureaux 3D . . . 55
MaW3D . . . 56
3DWM . . . 57
Task Gallery . . . 58

**3 Techniques et technologies 61**

3.1 Introduction du chapitre . . . 62

3.2 Interfaces et dimensions . . . 63
3.2.1 2D - Interfaces graphiques . . . 64
Redimensionnement automatique des fenêtres . . . 64
Bureaux multiples . . . 65
2D Rooms . . . 65
Emotional Icons . . . 66
Graphes dynamiques . . . 66
3.2.2 2.5D - Interfaces zoomables . . . 67
Point d'intérêt et contexte . . . 67
Fish-eye Views . . . 68
Les arbres hyperboliques . . . 69
La loupe à documents . . . 69
Les vues fractales . . . 71
Zoom et filtrage . . . 72
PAD++ . . . 72
Filtering Fish-eye Views . . . 73
Widgets de visualisation . . . 74
Les filtres . . . 74
Les loupes magiques . . . 75
3.2.3 Interfaces 3D . . . 75
Graphiques 3D . . . 77
Cityscapes . . . 78
Espaces de Benedikt . . . 79
Perspective Walls . . . 80
Cones Trees et Cam Trees . . . 81
Visualisation sphérique . . . 82
Salles 3D . . . 83
Information Cube . . . 84
*Widgets* de navigation . . . 85
Exemples . . . 85
3.3 Avancées technologiques . . . 86
3.3.1 Architecture des systèmes 3D . . . 87

Couche Accélération Matérielle . . . 88
Couche Interface Logicielle . . . 90
Couche Emulation . . . 91
Couche Application . . . 92
3.3.2 Normes et standards 3D . . . 93
Les API 3D . . . 94
OpenGL . . . 94
Direct3D . . . 98
Java3D . . . 100
Farhenheit . . . 103
Les moteurs 3D pour Internet . . . 105
VRML . . . 106
MPEG4 . . . 108
Metastream . . . 110
Scol et Blaxxun . . . 111
Virtools Dev . . . 112
3.3.3 Interactions . . . 114
Implémentation usuelle . . . 115
Les comportements . . . 118
3.4 Evolution du matériel . . . 119
3.4.1 Dispositifs d'entrée/sortie . . . 119
Périphériques de sortie . . . 119
Dispositifs de pointage . . . 120
Interactions 3D avec des dispositifs 2D . . . 121
Interactions 3D avec des dispositifs 3D . . . 122
3.4.2 Spatialisation du son . . . 124
Principes généraux . . . 125
Les deux techniques de synthèses sonores 3D . . . 126
Le modèle physique . . . 126
Le modèle psycho-acoustique . . . 127

**4 Expérimentations** **129**

4.1 Introduction du chapitre . . . 130

4.2 La toute première étude . . . 130
4.3 Les plateformes d'expérimentation . . . 133
4.3.1 Le langage VRML et son interpétation . . . 135
4.3.2 Le rendu graphique par OpenGL . . . 136
4.3.3 Spécificités de nos navigateurs VRML . . . 136
Manipulation de la caméra . . . 139
Aides à la navigation . . . 140
4.4 Bibliothèques numériques . . . 142
4.4.1 Contexte de cette étude . . . 143
L'ABU . . . 143
Le CNUM . . . 145
4.4.2 Les interfaces usuelles . . . 146
4.4.3 Les heuristiques visuelles . . . 147
4.4.4 Les interfaces 3D . . . 148
L'approche "réalité virtuelle" . . . 150
L'approche "réalité augmentée" . . . 150
4.4.5 Interface de recherche . . . 151
Classement par groupes . . . 151
Spatialisation des groupes . . . 153
Dans un atomium ou un cylindre . . . 153
Dans une sphère . . . 153
4.4.6 Interface de lecture . . . 156
Première génération en VRML . . . 156
Les choix généraux . . . 157
Détails d'implémentation . . . 158
Extension pour la vidéo . . . 160
Critiques de l'interface . . . 162
Deuxième génération avec Virtools Dev . . . 163
Le livre . . . 164
Le trépied . . . 165
Le sol et l'arrière plan . . . 167
Le panier de sélection . . . 168

4.4.7 La liaison entre les deux interfaces . . . 169
La métaphore du “cockpit” . . . 170
La métaphore du “camembert” . . . 171
La métaphore “sans dessus-dessous” . . . 172
4.4.8 Suite à donner à ces études . . . 172
4.5 Bureau 3D . . . 174
4.5.1 Origine et besoins . . . 175
4.5.2 Les travaux proches . . . 176
4.5.3 Description de notre projet . . . 178
4.5.4 Développements futurs . . . 178
4.6 Son spatialisé . . . 180
4.6.1 Choix de la méthode de rendu . . . 180
4.6.2 Choix du langage . . . 182
MPEG4 . . . 183
VRML . . . 184

**5 Critiques et solutions proposées 187**
5.1 Introduction du chapitre . . . 188
5.2 Enseignements tirés de nos expérimentations . . . 189
5.2.1 Critiques des choix de conception . . . 189
Naïveté des interfaces . . . 189
Intérêt pour la 3D décrite . . . 190
5.2.2 Insuffisances de VRML pour nos interactions . . . 191
Création/Destruction de contenu . . . 192
Immobilité des objets . . . 194
Rangement des objets . . . 194
Changement de textures . . . 196
5.2.3 Insuffisances de VRML pour les scènes sonores . . . 197
Ce que VRML peut faire . . . 197
Ce que VRML ne sait pas faire . . . 199
5.3 Extensions du langage VRML . . . 199
5.3.1 Comportements . . . 200
Comportements Vs Composants . . . 202

Extraction de comportements . . . . . 204
Les comportements 2D . . . . . 207
Les comportements 3D . . . . . 209
Description des comportements . . . . . 211
5.3.2 Son . . . . . 212
Extensions . . . . . 213
Sources Sonores . . . . . 213
Effets de salle . . . . . 214
Filtrages sonores . . . . . 215
Implémentation . . . . . 217

**6 Conclusion et travaux futurs** **221**

**Publications** **231**

**Références** **233**

**Liste des acronymes** **251**

**Liste des figures** **253**

**Liste des tableaux** **257**

# Chapitre 1

# *Introduction*

## Les interfaces informatiques

En informatique, on entend par "interface", une couche spécialisée dont le rôle est de présenter sur un écran des informations à un utilisateur et ceci dans le sens le plus général que peut prendre le terme "information" puisqu'il peut s'agir d'informations visuelles (comme dans un jeu ou un modeleur), de données provenant d'une base de données, ou de toute autre information abstraite ou concrète qu'un utilisateur manipule par l'intermédiaire d'un logiciel. Quelle que soit l'utilité d'une interface, elle doit, d'une part, offrir une organisation intelligente de l'espace de visualisation disponible et, d'autre part, utiliser une représentation compréhensible et adaptée au type d'informations. Pour cette dernière propriété, nous parlerons de métaphores pour les représentations ressemblant même de loin à un élément du monde physique.

D'après le paragraphe précédent, on pourrait comprendre que l'IHM (Interaction Homme-Machine) concerne uniquement la visualisation de données. Cependant, le domaine couvert par cette discipline est beaucoup plus large et particulièrement depuis que la représentation des informations sur écran est passée du mode textuel monochrome au mode graphique couleur. Ce changement a, bien sûr, permis d'accroître les possibilités offertes pour la représentation des informations mais le grand bouleversement est également venu des périphériques d'entrée associés (comme le stylo optique, la souris, la reconnaissance vocale ...) qui ont introduits de nouveaux et riches moyens d'interagir avec un ordinateur par rapport à l'utilisation unique d'un clavier.

Depuis lors, la définition informatique du terme "interface" dans le vocabulaire du domaine de l'IHM s'est rapprochée de la définition faite pour ce mot dans la discipline de laquelle il

provient : la chimie[1]. En informatique, on saisit bien que la présentation des informations et leurs réactions aux stimuli de l'utilisateur forment justement ce dispositif qui autorise les échanges entre l'homme et l'ordinateur. Les interfaces homme-machine revêtent donc deux aspects différents auxquels il faut apporter une égale attention, sans quoi elles ne permettraient pas d'échanges bilatéraux. Le premier de ceux-ci, présenté précédemment, concerne leur apparence qui est liée à la technique de visualisation utilisée et aux objets la composant (comme les menus, les boutons, ...). Le deuxième aspect, quant à lui, s'attache aux interactions possibles entre l'utilisateur et l'interface et, plus particulièrement, avec les éléments la composant. Le rassemblement de ces deux aspects nous conduit à considérer l'IHM non pas seulement comme l'étude des interfaces homme-machine mais également comme l'étude des interactions entre l'homme et la machine.

## Les interfaces métaphoriques

Considérant également les interactions possibles avec une interface, et les informations qu'elle présente, de nouvelles fonctionnalités doivent être introduites pour gérer efficacement ce nouvel aspect. On parlera également de métaphores pour les interactions car la tendance actuelle est d'essayer de les rendre intuitives et proches d'un geste réel et cela est grandement facilité par l'apparition de dispositifs d'entrée comme le gant de réalité virtuelle ou les pointeurs et capteurs 3D.

Bien qu'évoquant des périphériques qui paraissent surgir du futur (pour un non spécialiste) et promettre une progression rapide des IHM, le sursaut n'a toujours pas eu lieu. Il suffit pour cela de constater que les gants et casques virtuels ainsi que la plupart des capteurs existent depuis longtemps mais qu'ils ne sont pas encore largement utilisés [Fuchs99, Fuchs02]. Leur sont toujours préférés la souris et le clavier. De son côté, l'interface graphique "moderne", mise au point dans les années 70 au centre de recherche de Xerox PARC à Palo Alto, est toujours utilisée à l'heure actuelle. Cependant, elle semble aujourd'hui atteindre ses limites pour présenter les nombreuses données qu'un système d'informations est capable de stocker et de manipuler. Cette interface composée d'éléments tels que les fenêtres, les boutons, le pointeur de souris, les icônes, les menus, paraît en effet incapable de fournir une surface d'affichage suffisante pour la visualisation et, par extension, pour l'interaction avec ces nombreuses données. La tendance est même de réduire encore plus l'espace de travail par la multiplication des barres de boutons, des barres d'aides et autres gadgets qui visent à fournir aux utilisateurs de nouveaux outils pour l'interaction avec les données.

## Limitations des interfaces actuelles

Le problème des interfaces actuelles vient principalement de la façon dont est utilisée la surface de visualisation réduite offerte. L'usage des interfaces est fondé sur le principe de la pile, d'un monticule de documents dont seuls les premiers sont visibles, quelques fois même seulement partiellement. En effet les fenêtres opaques des différentes applications doivent

[1] définition du mot "interface" en chimie : dispositif qui permet d'opérer des échanges bilatéraux entre deux éléments.

être superposées les unes sur les autres à cause de l'impossibilité de changer de facteur de zoom ou de position du point de vue. Or, la multiplication des applications ouvertes et donc du nombre de fenêtres affichées complexifie les tâches d'organisation, de localisation et de récupération des données jusqu'au point où cette interface n'est plus utilisable.

Le but premier d'une interface (aussi bien dans le contexte de la visualisation de documents et d'informations en 3D que de manière plus générale) est de réduire, par l'utilisation de techniques graphiques avancées, le coût induit par la recherche et la compréhension d'une information [Robertson93a]. Pour aller dans ce sens, les métaphores sont maintenant pensées pour que le plus grand nombre de données soit assimilable par le système perceptif de l'utilisateur (dont le temps de réponse est de l'ordre de 10 à 100 millisecondes) plutôt que par son système cognitif (dont le temps de réponse va de quelques centaines de millisecondes à quelques minutes pour des données fortement dé-corrélées de leur représentation graphique). La conception de métaphores, s'appuyant sur la perception, peut donc accélérer le traitement des informations par l'utilisateur de façon non négligeable.

## Plan de la thèse

Cette thèse se propose, dans un premier temps, d'aborder les différents aspects qui ont, ensemble, participés à l'évolution des "paysages informationnels" (c'est-à-dire de l'IHM appliquée à la visualisation de nombreuses informations et à la manipulation de documents). Les éléments entrant en jeu dans cette évolution sont nombreux et le choix a été fait de les présenter par couche, en partant de la couche application et en descendant jusqu'à la couche matérielle. Il paraissait en effet difficile de parler des travaux sur la visualisation de données sans les lier avec les différentes avancées technologiques. Notamment, les possibilités graphiques, spécialement en 3D, des ordinateurs personnels permettent d'expérimenter des interfaces qui ne pouvaient pas être mises en œuvre il y a encore quelques années. Un autre exemple de liaison entre le domaine de la visualisation de documents et les technologies est le *World Wide Web* qui autorise aujourd'hui la récupération et la diffusion de nombreux médias.

Nous avons fait le choix de séparer les aspects applicatifs et technologiques en deux états de l'art différents. Dans le chapitre 2, nous définissons, étudions et présentons les problèmes posés par la visualisation de larges collections d'informations. Nous parcourrons ensuite différents projets de visualisation s'attachant à divers domaines. Le chapitre 3 se propose d'étudier les différentes solutions techniques et leurs atouts pour apporter une solution à ces problèmes particuliers. En pratique, les interfaces, particulièrement les interfaces 3D, reposent sur des normes et standards que nous détaillerons également. Au niveau matériel, nous abordons rapidement les évolutions des stations graphiques (cartes graphiques et microprocesseurs) et celles des dispositifs de pointage qui ont permis de faire progresser le domaine.

Nous décrivons ensuite, dans le chapitre 4, nos expérimentations qui touchent principalement à trois domaines : les bibliothèques numériques, les bureaux 3D et le son spatialisé. Pour les premières, une attention particulière a été portée sur la présentation de la collection (proche de la visualisation d'informations) que sur la métaphore du poste de lecture (proche de la visualisation de documents). Pour le bureau 3D, le challenge était de proposer

une solution originale pour organiser un espace de travail encombré d'applications. Enfin, nous avons étudié le problème de la spatialisation sonore avec VRML dans le cadre d'un projet artistique initié par Cécile Le Prado.

Le chapitre 5 présente les enseignements que nous pouvons tirer des différentes expérimentations. En particulier, nous soulignons les lacunes du langage VRML pour décrire des applications audiovisuelles 3D interactives. Pour solutionner ce problème, nous proposons nos propres extensions sous forme de nouveaux nœuds comportementaux et sonores. Ces nœuds complètent le modèle de description géométrique de VRML et permettent de limiter la part du code à adjoindre à une scène pour la rendre interactive. Le dernier chapitre présente les conclusions de ces futurs, leurs perspectives ainsi que les travaux futurs.

# Chapitre 2

# *Paysages informationnels*

**Contenu du chapitre**

*Ce chapitre présente les "paysages informationnels" par les tâches qui leur incombent et par quelques exemples classés par domaines. La liste des projets décrits succinctement dans ce chapitre n'est absolument pas exhaustive. Elle rassemble les principaux projets afin de souligner l'importance que prend, depuis quelques temps, la visualisation de données et de documents, en particulier à l'aide d'interfaces 3D.*

*Ce chapitre est organisé de la manière suivante :*


- Section 2.1 – Introduction du chapitre – p. 6 : *Explications brèves pour marquer la nécessité de rechercher des nouvelles visualisations pour gérer les nombreuses données disponibles.*
- Section 2.2 – Les tâches couvertes par les paysages informationnels – p. 8 : *Description des différentes tâches auxquelles un système gérant de multiples données doit répondre.*
- Section 2.3 – Les différents domaines d'application – p. 16 : *Présentation de quelques projets de visualisation classés par domaines.*
- Section 2.4 – Autour de nos préoccupations – p. 48 : *Présentation des projets de visualisation pour les bibliothèques numériques et les environnements applicatifs 3D.*

## 2.1 Introduction du chapitre

L'évolution rapide de la technologie et la baisse du coût du matériel informatique a provoqué une explosion de la puissance de calcul et une informatisation galopante des ménages. Les systèmes informatiques connaissent régulièrement une augmentation de leurs caractéristiques, aussi bien du point de vue de la puissance de calcul, que du point de vue capacité de stockage. Leurs prix chutent rapidement de telle sorte que cette puissance de traitement devient abordable pour le grand public. Cette prolifération de machines performantes, couplée aux avancées technologiques des réseaux et l'avènement du *World Wide Web* ont causé une sorte de "big bang" informationnel.

Malheureusement, "information" n'est pas le terme correct pour décrire l'abondance de données. On devrait plutôt employer le terme "bruit". En effet, l'ensemble des données disponibles et accessibles à l'heure actuelle a grossi en des proportions gargantuesques alors que la croissance correspondante en nombre d'informations est beaucoup moindre. Il en résulte de nouveaux problèmes. Précédemment, le problème résidait en l'absence ou en l'impossibilité de trouver une information particulière à cause du manque d'indexation, de bases de données spécialisées voire du manque d'informations tout court. Aujourd'hui, le problème s'est déplacé car l'abondance de données fait que l'information souhaitée existe probablement. Cependant, parmi la multitude de résultats qui nous sont soumis après une requête, la difficulté réside en trouver, trier, organiser, comprendre et retrouver l'information pertinente. Ces tâches d'aide à la récupération de données sont regroupées sous le terme "extraction de données" (*data mining*) qui illustre parfaitement la nécessité d'explorer, de creuser parmi les données.

Alors que l'on parle d'autoroutes de l'information, le travail de certains chercheurs est de trouver différents outils et techniques pour donner aux utilisateurs le plus confortable et le plus sûr des cheminements pour arriver au plus vite et au moindre coût aux informations souhaitées. Les algorithmes et les méthodes pour le *data mining* sont développés pour automatiser le filtrage des informations : produisant des système de recherche "intelligents" qui peuvent retourner des données plus pertinentes aux utilisateurs. Par ailleurs, l'attention portée aux systèmes de visualisation est de plus en plus grande. Ces systèmes cherchent à exploiter les qualités cognitives, perceptives et intuitives des utilisateurs pour les aider à trouver des informations qui pourraient être oubliées par un système de recherche automatisé.

Cette manière d'appréhender les informations est à l'origine de l'expression "paysages informationnels" qui est une métaphore signifiant tout simplement que l'utilisateur est immergé dans une multitude de données et qu'il a la possibilité de les explorer[1]. Tout comme une personne pourra s'imprégner d'un paysage en le balayant depuis un point de vue dégagé, dans le cas de paysages informationnels, l'utilisateur disposera d'une position de choix pour observer à sa guise les données informatiques. La comparaison ne s'arrête cependant pas uniquement à cette position privilégiée du point d'observation. Dans les deux cas, cette place de choix induit la possibilité de voir un plus grand nombre de choses. Mais encore, tout comme un touriste admirant un magnifique paysage depuis un belvédère, pourra observer plus en détail quelques zones grâce à des jumelles, l'utilisateur face à un paysage

[1] C'est Muriel Cooper, en 1979, qui utilisa pour la première fois l'expression "paysages informationnels".

informationnel disposera d'outils pour effectuer le même genre de tâche : voir plus en détails les données. Dans les deux cas, l'observateur du paysage pourra, à sa guise, passer d'une vue d'ensemble à une vue rapprochée sur un détail sans pour autant perdre le contexte (l'environnement) dans lequel se trouve ce point d'intérêt. Le meilleur exemple pour illustrer cette comparaison sont les *cityscapes* (paysages urbains - voir page 78), utilisant des interfaces s'inspirant des *landscapes* (paysages) dans lesquelles l'utilisateur se promène dans une "ville de données".

Les études sur l'utilisation des interfaces pour la visualisation d'informations sont nombreuses. Ces travaux deviennent d'une aussi grande importance (du point de vue de leur nombre aussi bien que pour promouvoir l'utilisation des interfaces) que ceux portant sur les aspects purement graphiques. Sans les appliquer à un type précis d'interfaces, nous tentons ici d'introduire les problèmes liés à la manipulation de données et de présenter les résultats importants des différentes recherches sur ce sujet. Les diverses techniques employées pour construire des interfaces capables d'afficher un grand nombre de données seront quant à elles vues dans la section 3.2 à partir de la page 63.

On peut imaginer les interfaces graphiques comme une photographie, une prise de vue sur les données, comme si l'utilisateur regardait les informations à travers un appareil photographique. Pour les interfaces classiques 2D, cette caméra est fixe et placée à la verticale de la surface rectangulaire délimitant le bureau. Lorsque nous parlons d'agrandir ou d'étirer le volume de visualisation, nous voulons dire : mettre en place un mécanisme permettant d'élargir cette surface rectangulaire. La plupart des solutions apportées ces dernières années mettent en place une interface 2D augmentée ou interface 2.5D où la caméra, par laquelle sont visualisées les données, peut se déplacer en profondeur ou dans lesquelles les données sont dessinées avec plus ou moins de détails en fonction de leur distance à un point d'intérêt.

Ces interfaces 2.5D reprennent bon nombre des avantages de la 3D. Les interfaces zoomables, en particulier, reprennent la gestion dynamique du point d'intérêt et des informations contextuelles par l'utilisateur lorsqu'il modifie son point de vue et cela sans calculs supplémentaires nécessaires pour l'affichage du contexte. Cependant, la conception d'une interface 2.5D n'est plus une problématique de recherche mais plutôt une alternative technologique réaliste face à la trop grande puissance de calculs requise à l'heure actuelle pour la 3D. En effet, dans un avenir proche, le pourcentage d'occupation du microprocesseur nécessaire pour afficher une scène 3D importante sera comparable aux ressources nécessaires aux microprocesseurs actuels pour gérer une interface 2.5D.

Cette partie décrit une variété de techniques de visualisation d'informations en 3D et de systèmes de recherches qui ont été développés pour faciliter la compréhension de grands systèmes de données. Nous avions quelques interrogations quant à la manière de présenter ces différentes techniques. Nous avons opté finalement pour un classement par type d'interface utilisée (2D, 2.5D ou 3D) dans la section 3.2 page 63, en restant très générique. Nous avons néanmoins effectué tout d'abord une présentation par tâche (trouver, appréhender le contexte et le contenu, organiser et retrouver) dans la section suivante mais aussi par type d'applications (2.3 page 16) pour avoir une idée plus précise des projets et des domaines couverts par les techniques de visualisation. Cette classification nous a été inspirée par l'ouvrage de C. Chen sur la visualisation d'informations et les environnements virtuels

[Chen99]. Quoi qu'il en soit, quelle que soit la manière de classer ces techniques, le but est ici de montrer les techniques actuellement disponibles pour la visualisation de documents. Pour chaque prototype ou système de visualisation ou de recherche d'informations, nous donnons un bref aperçu de leurs fonctionnalités et capacités.

Avant de rentrer dans le vif du sujet, en parlant des domaines d'applications et des différentes approches techniques, nous voudrions introduire les tâches auxquelles les "paysages informationnels" tentent d'apporter une solution efficace.

## 2.2 Les tâches couvertes par les paysages informationnels

Bien évidemment, la première utilité des paysages informationnels est de présenter des données à l'utilisateur. Cependant, ces deux termes accolés prennent tout leur sens lorsque les interfaces proposées facilitent la structuration et l'exploration et avant tout la compréhension de grands ensembles de données. La présentation de nombreuses données est en effet devenue un problème majeur. Alors qu'il y a encore très peu de temps, ce problème n'était entrevu que pour l'affichage des résultats de requêtes pour les bases de données, il apparaît aujourd'hui dans différents cas dont nous parlerons dans les sections 2.3 page 16 et 2.4 page 48.

Ces nouveaux domaines dans lesquels l'étude des interfaces a pris une importance toute particulière, sont tous caractérisés par deux facteurs principaux : l'augmentation constante des capacités de stockage et de traitement des ordinateurs et la croissance explosive du *World Wide Web*. Ces deux paramètres font, en effet, que de plus en plus de données sont recherchées, manipulées, lues, traitées et archivées. Non seulement le nombre de données stockées a augmenté considérablement, mais également, le nombre de traitements possibles, en un temps fixe, sur celles-ci. Or, la visualisation de ces ensembles croissants (aussi bien les données elles-mêmes que les traitements) se heurte au problème de l'espace réduit de l'écran. Les études sur les paysages informationnels proposent donc d'offrir à la fois une solution face à l'essoufflement des interfaces WIMP[2] et de mettre en évidence des propriétés pour assister l'utilisateur dans les différentes tâches qu'il peut opérer sur un ensemble de données.

La caractérisation, faite par Laurent Robert dans sa thèse [Robert01], des tâches de haut niveau associées à l'exploration de documents hypermédias est un bon point de départ pour illustrer l'importance des paysages informationnels. Il énumère ces différentes tâches et les caractérise de la manière décrite dans la table 2.1.

La tâche "Comprendre" telle qu'elle est définit par L. Robert prend véritablement tout son sens pour les documents hypertextes puisque les liaisons entre documents sont des informations à part entière. Mise à part l'analyse du contenu, les autres définitions de cette tâche sont relatives au contexte dans lequel se trouve un document : contexte donné par les hyperliens présents dans le document. Pour que cette classification s'adapte à tout type de

[2] *Windows Icons Menus Pointer* : interfaces classiques constituées de fenêtres, icônes et menus manipulés par l'intermédiaire du pointeur de la souris.

| Tâche | *Description* |
|---|---|
| **Trouver** | Accéder aux documents recherchés (en fonction d'un objectif, d'un centre d'intérêt, etc.). |
| **Comprendre** | Identifier les relations sémantiques et/ou structurelles entre les documents, faire des recoupements, associer des idées, analyser le contenu, etc. |
| **Organiser** | Etablir un schéma de classification personnalisé des documents jugés intéressants, après les avoir collectés. |
| **Retrouver** | Accéder une nouvelle fois, plus simplement et plus rapidement, à des documents préalablement consultés. |

**Tab. 2.1** Les tâches de haut niveau décrites par L. Robert [Robert01].

données, nous préférons scinder "Comprendre" en "Appréhender le contenu" et "Appréhender le contexte" pour obtenir respectivement une tâche de récupération des informations lisibles dans le document et une seconde relative au positionnement de ce document par rapport aux autres. Ces deux tâches de compréhension, associées à deux types d'informations différentes (le contenu et le contexte), semblent nécessaires pour caractériser plus clairement la visualisation d'informations et la visualisation de documents. Pour la première, la structure des données est prépondérante d'où l'importance du contexte. Pour la seconde, le point d'intérêt central est le document, d'où l'importance de la lisibilité et de la compréhension de son contenu.

### 2.2.1 Trouver

Cette tâche concerne exclusivement la recherche des documents. Cette recherche peut être opérée soit, de manière classique, par un formulaire de requête, soit par une exploration des documents. Dans le deuxième cas, cette tâche est fortement associée à la visualisation. Tous les systèmes de visualisation que nous verrons par la suite, dans les exemples ou dans les classifications, répondent à cette problématique d'aide à l'exploration mais aussi d'organisation des documents. Tous semblent s'accorder sur l'importance de la navigation interactive et de la fluidité de celle-ci pour mettre en avant le système perceptif de l'utilisateur. De telle sorte que l'on entend de plus en plus parler de "perception d'informations" plutôt que d'exploration d'informations.

Les heuristiques visuelles, rendues possibles par de tels systèmes temps-réel exploitant des techniques graphiques avancées, complètent parfaitement les recherches classiques. Dès lors qu'une grande majorité des documents sont présentés dans l'espace informationnel et que leurs caractéristiques sont discriminées (par leur taille, par leur couleur, par leur

texture, etc.), l'utilisateur peut découvrir des informations pertinentes et certaines de leurs caractéristiques par un simple coup d'œil.

Pour éviter le phénomène de désorientation et d'égarement dans la masse d'informations lors de l'exploration d'espaces hypertextes, et plus généralement de paysages informationnels, il est nécessaire d'offrir aux utilisateurs des outils d'aide à l'exploration [Bernstein91, Nielsen90, Rivlin94, Utting89]. Ceux-ci aident non seulement à trouver plus rapidement les informations recherchées mais aussi à comprendre le contexte dans lequel un document s'inscrit. Les outils d'exploration sont nombreux et quelle que soit la technologie employée pour les implémenter, ils se classent parmi les catégories de la table 2.2.

Les visualisations proposent généralement un certain nombre de ces outils déclinés pour convenir au type d'interface utilisée. C'est notamment le cas des arbres hyperboliques [Lamping95] ou des vues avec distorsion [Furnas86, Robertson93b] en 2.5D qui proposent une carte des données. En 3D, les arbres coniques [Robertson91] proposent également une telle visualisation du contexte. Le *perspective wall* [Mackinlay91], habituellement utilisé pour parcourir les données structurées séquentiellement, peut être une manière d'implémenter une sorte de visite guidée.

Pour la tâche "Trouver", la mise à disposition de représentations associant une vue globale à une vue locale, quelque soit l'outil utilisé pour l'exploration, permet de profiter du sens de l'observation de l'utilisateur. En plus de l'aide apportée par les outils précédents, dans une interface liant contexte et point d'intérêt, il est possible de découvrir des informations intéressantes par simple observation, reconnaissance de motifs, de couleurs ou d'autres caractéristiques graphiques. Grâce à leur grand espace disponible pour organiser les données, les visualisations 3D sont particulièrement bien adaptées pour implémenter à la fois les heuristiques de recherche classique et les heuristiques visuelles.

### 2.2.2 Appréhender le contenu

Le premier but d'une interface documentaire est de présenter correctement les données afin que l'utilisateur puisse les lire. Cela suppose d'utiliser des techniques adéquates pour améliorer la lisibilité et pour obtenir rapidement les informations essentielles qui caractérisent un document. Pour cela, un système de visualisation devra proposer des solutions efficaces pour offrir les services décrits dans le tableau 2.3. Ces services valent aussi bien pour la présentation générale des données (l'affichage de la structure globale de la base de connaissance) que pour l'affichage du contenu d'un document.

### 2.2.3 Appréhender le contexte

Il est maintenant largement reconnu que la compréhension de la structure globale des données présentées est essentielle à la fois pour s'y déplacer, y trouver l'information désirée et la replacer parmi les autres données (son contexte) [Furnas86, Fairchild88, Mackinlay91, Ware93]. Nous pensons également que la présentation du contexte offre une nouvelle heuristique visuelle de recherche [Cubaud98]. Sachant que le contexte d'un

| **Outils d'exploration** | *Description* |
|---|---|
| **Traditionnels** | Ces outils, que nous avons pris l'habitude de manipuler lors de la consultation de livres, sont les catalogues, les tables des matières, les index, les résumés, les glossaires. |
| **Points de repère** | Ce sont des informations particulières jugées comme étant le point de départ de la consultation. Ce sont par exemple les *home pages* des sites *web*. Ces pages sont le point d'entrée d'un site et permettent d'accéder, directement ou non, à toutes les informations présentes dans le site. |
| **Visite guidée** | Ce mécanisme permet de définir un parcours que les utilisateurs suivent pour avoir un aperçu du contenu d'un espace informationnel. Dans le cas d'inforations structurées sous la forme de graphes, une visite guidée offre aux utilisateurs novices un parcours séquentiel pour explorer les données. |
| **Cheminement** | La mémorisation du chemin emprunté par l'utilisateur lui permet à tout moment de revenir en arrière s'il s'est "perdu" ou "égaré". Ces historiques de navigation, rencontrés sous différentes formes, textuelles ou graphiques, peuvent incorporer des mécanismes de classement selon le contenu ou la position du document dans l'espace informationnel afin de présenter une carte plutôt qu'une liste chronologique des déplacements. |
| **Marquage** | Cet outil permet à l'utilisateur de sauvegarder l'emplacement d'un document particulier et de le désigner comme un point de repère. L'ensemble de ces marques peut être présenté sous forme de liste ou, si l'utilisateur a effectué un classement, sous forme d'arbres. |
| **Cartographie** | Cet outil, tout comme certains historiques de navigation, présente à l'utilisateur la carte d'un espace informationnel depuis laquelle il pourra accéder directement aux documents. Il pourra aussi plus facilement comprendre la structuration globale de l'environnement et identifier le contexte dans lequel se trouve un document d'intérêt. |

**Tab. 2.2** Les outils d'aide à l'exploration.

document d'intérêt est souvent construit de telle manière que les documents y figurant ont un contenu semblable, l'utilisateur peut alors découvrir de nouvelles informations en regardant ce contexte.

La présentation et la compréhension du contexte permettent également, souvent, de mieux saisir la teneur du contenu d'un document d'intérêt. Pour faciliter l'extraction intuitive

| **Services** | *Description* |
|---|---|
| **Représentation** | La manière dont les éléments d'une interface sont représentés, les métaphores utilisées, les caractéristiques visuelles. |
| **Abstraction** | Ce qui doit être présenté, le niveau de détails, la taille du contexte. |
| **Navigation** | Les paradigmes d'exploration fournis pour étudier les informations. |
| **Corrélation** | La liaison mentale entre les données à explorer et l'abstraction visuelle proposée. |

**Tab. 2.3** Les services pour accéder aux données d'un système de visualisation.

des informations, les différents systèmes et techniques que nous présentons par la suite regroupent géographiquement les documents qui ont des caractéristiques proches. Ces liaisons entre documents respectent selon les cas, la proximité sémantique, temporelle (ou sur toute autre propriété), ou encore, quand elle existe, la structuration globale liant tous les documents (dans le cas de documents hypertextes par exemple).

Comme pour sa construction, l'affichage du contexte peut suivre plusieurs règles, parmi lesquelles le type d'abstraction proposé pour représenter les éléments de l'interface. La table 2.4 présente les trois types d'interfaces que l'on peut rencontrer.

| **Type d'interfaces** | *Définition* |
|---|---|
| **Interfaces techniques** | Elles s'adressent plutôt aux spécialistes comprenant le modèle de données sous-jacent (pour la construction de requêtes vers un SGBD, pour la visualisation de processus). |
| **Interfaces métaphoriques** | Les objets présents dans ce type d'interfaces sont des objets réels avec lesquels, par conséquent, les interactions correspondent à des actions réelles. |
| **Interfaces "idiomatiques"** | Les objets et les interactions qui y sont proposés ne sont pas nécessairement issus du monde réel. Mais ces interfaces offrent des métaphores qui ne s'oublient pas (comme une langue maternelle d'où le terme "idiome"). Un exemple est l'icône dont on ne connaît pas *a priori* le fonctionnement et qui ne nécessite qu'un seul apprentissage. |

**Tab. 2.4** Les trois types d'interfaces.

Les interfaces "idiomatiques", que nous préférons, se caractérisent par l'utilisation efficace, ou tout du moins bénéfique, d'éléments graphiques. La volonté de trouver des objets métaphoriques pousse trop souvent les créateurs d'interfaces à fournir également des déplacements métaphoriques qui sont parfois inutiles et incommodes pour les utilisateurs. En poussant à l'extrême cette volonté de trouver une interface métaphorique, nous rencontrons les interfaces de réalité virtuelle utilisant uniquement ce paradigme comme [Kipp97] et [Celentano99]. Celles-ci ne nous convainquent ni ne nous conviennent à cause de l'emploi superflu et, en général, sans valeur ajoutée de la 3D car elles ne s'affranchissent pas de certains aspects qui nous limitent dans le monde réel et qui n'ont pas de raison d'exister dans une interface 3D (comme la gravité ou les constructions pour se protéger des intempéries). Ces projets de réalité virtuelle n'ont menés qu'à la production de belles images et c'est une forme d'utilisation de la 3D que regrette P. Young [Young96].

### 2.2.4 Organiser

Un problème crucial lorsque l'on crée un paysage informationnel est de trouver une correspondance (un *mapping*) efficace entre les données et leur représentation et leur localisation dans l'environnement virtuel. La condition *sine qua none* à respecter pour s'assurer de l'efficacité d'un paysage (qui n'est pas un problème uniquement lié à la 3D) est de créer une configuration spatiale dans laquelle les données et les relations peuvent être aisément interprétées en ne prenant en compte que leur position et leur représentation. En ce qui concerne la construction des "paysages informationnels peuplés"[3]), quatre approches différentes sont présentées dans les paragraphes suivants puis comparées [Benford94a, Benford94b, Benford97, Colebourne94].

#### Cyberspace de Benedikt

Comme nous le verrons page 79 dans ce rapport de thèse, le Cyberespace de Benedikt est créé en "mappant" des attributs des données sur les dimensions intrinsèques (les propriétés des objets) et extrinsèques (liées à la localisation ou à la taille des objets et plus généralement à la métrique des objets) des objets 3D correspondants. Cette méthode nécessite une spécification claire de la base de données à visualiser afin de déterminer quelles propriétés des données doivent ou peuvent devenir des attributs des dimensions intrinsèques et extrinsèques de la représentation visuelle. Q-PIT est un exemple d'une telle approche et est décrit page 20. Un autre exemple est *TripleSpace* [Mariani92], qui a été l'un des premiers systèmes à employer les espaces de Benedikt.

#### Partitionnement statistique et mesures de proximité

Des méthodes statistiques peuvent être utilisées pour analyser les contenus d'une base de données afin de grouper les éléments selon leur proximité sémantique. Par exemple, les éléments d'un fonds documentaire peuvent être groupés en fonction des mots-clés qui les

[3] PITs = *Populated Information Terrains.*

caractérisent. De telles analyses fournissent en général des résultats sous forme de pourcentages de rapprochement entre documents qui peuvent alors être utilisés pour effectuer une correspondance appropriée dans l'espace de Benedikt. Plus les objets sont sémantiquement liés, plus ils seront proches géographiquement dans l'environnement virtuel. Des systèmes qui optent pour cette approche sont par exemple VIBE [Olsen93] et BEAD [Chalmers92]. L'idée originale de VIBE a été ensuite prolongée et développée en utilisant une interface 3D pour produire VR-VIBE [Benford95]. Nous verrons ces systèmes de visualisation à partir de la page 20

### Hyperstructures

Les hyperstructures sont créées depuis des bases de données des objets ayant un nombre quelconque de liens explicites entre eux. L'exemple le plus simple et répandu d'hyperstructures est la structure d'un ensemble de documents HTML. Un autre exemple est le graphe de dépendance formé par les composants d'un système logiciel, comme les fonctions. Ces structures sont généralement visualisées en étendant à la 3D les algorithmes standards de dessin de graphes, et en utilisant des représentations spécialisées en 2.5D ou 3D telles que les *fish-eye views*, les arbres coniques ou les murs en perspective.

De nombreux projets ont mis en œuvre des algorithmes pour organiser automatiquement les graphes. L'article d'Herman, Melançon et Marshall [Herman00] décrit les différentes dispositions (*layouts*) rencontrées pour visualiser les graphes, aussi bien en 2D, qu'en 2.5D ou qu'en 3D. Cet article présente également les interactions et les navigations implémentées dans quelques projets fondateurs comme les *Fisheye Views* de G. W. Furnas [Furnas86] ou les *Cone Trees* de J. D. Mackinlay *et al.* [Mackinlay91]. D'autres excellents articles ou livres de synthèses sont disponibles sur les techniques et algorithmes d'affichage de structures arborescentes [Battista94, Battista99].

### Approches centrées sur l'homme

Cette approche n'est pas appropriée pour la génération automatique des visualisations mais s'appuie plutôt sur l'utilisateur ou le concepteur du système pour représenter de manière appropriée les données à visualiser. Cette méthode se prête bien à l'utilisation de métaphores virtuelles (telles que les villes, les bâtiments et les salles) dans lesquelles l'utilisateur navigue plus naturellement au milieu des données immergées puisqu'il profite de ses acquis dans le monde réel. Cette approche est issue des premiers travaux initiés par Henderson sur la métaphore de salles [Henderson86]. Cependant, la laborieuse et difficile spécification de telles métaphores est le problème principal de cette approche. Créer un environnement virtuel, qui permet à la fois d'utiliser les expériences engrangées par les utilisateurs dans le monde réel, de lui fournir tout de même de nouveaux services, et de trouver une structuration et une représentation appropriées des données, n'est pas une tâche facile.

Il est tout à fait possible de combiner l'approche de Benedikt avec cette approche centrée sur l'utilisateur en lui laissant le choix du type de correspondances entre les attributs

et les dimensions. C'est, par exemple, ce que Gray *et al.* ont autorisé dans leur projet *Iconographer* [GRAY90].

### Comparaison et "applicabilité"

La différence essentielle entre ces quatre approches précédentes est la manière dont elles déterminent la localisation des données. L'approche de Benedikt le fait en fonction des valeurs de trois attributs pris parmi l'ensemble des attributs. Avec l'approche statistique, la spatialisation est opérée en calculant des taux de similitude entre objets et en effectuant des combinaisons pondérées de plusieurs valeurs d'attributs qui ont une relation avec d'autres objets. Les hyperstructures quant à elles, s'appuient sur la connaissance (spécifiée ou calculée) de liens qui indiquent la proximité sémantique des objets. Enfin, les utilisateurs choisissent le mode de représentation et de spatialisation dans l'approche centrée sur l'homme.

Les premières expérimentations faites sur ces différentes approches montrent que chacune s'adapte à un type particulier de données.

L'espace de Benedikt s'attache plus à des données qui sont naturellement ordonnées. Trier et classer des objets ne pose pas de problème avec les attributs numériques et pas plus avec les attributs textes qui sont ordonnés alphabétiquement. Mais cet ordre n'a de sens que s'il fournit une compréhension sémantique à l'utilisateur. Par exemple, une liste ordonnée de nom dans un agenda est utile pour la recherche alors que classer alphabétiquement des mots-clés de documents n'aide en rien. Le cyberespace de Benedikt pose également des problèmes lorsque les attributs ont des valeurs multiples. En résumé, cette technique d'organisation sera applicable à des collections d'objets très bien structurées et ordonnées comme c'est le cas de la plupart des bases de données relationnelles.

L'approche statistique, par opposition à la précédente, s'oriente plus vers des données textuelles moins bien structurées. En particulier, cette approche a la capacité d'extraire les relations sémantiques qui ne sont pas connues à l'avance et qui n'ont donc pas pu être intégrées dans un classement pré-établi.

Les visualisations d'hyperstructures s'intercalent entre les deux approches précédentes puisque les relations entre objets sont explicites mais *a priori* trop nombreuses pour être pré-établies. C'est le cas notamment des pages HTML qui peuvent référencer indirectement une quasi infinité de documents. Les domaines de prédilection de cette approche sont bien évidemment la visualisation des documents hypertextes mais aussi des systèmes de fichiers.

L'approche centrée sur l'humain présuppose le développement d'outils et de techniques pour spatialiser les informations en fonction des choix de l'utilisateur. De tels outils peuvent être utiles également aux trois approches précédentes. Il y a peu d'applications pour lesquelles une telle approche est développée mais il existe une grande diversité d'applications potentielles qui pourraient en bénéficier. Le développement de telles visualisations orientées vers l'utilisateur favorise la communication, le déplacement et les liaisons entre documents ; aspects qui n'ont été vus que conceptuellement jusqu'alors.

### 2.2.5 Retrouver

La possibilité de retrouver rapidement les documents fréquemment étudiés est particulièrement important pour un utilisateur. Il n'est pas envisageable de le forcer à recommencer une tâche de recherche, longue et fastidieuse, à chaque fois qu'il veut consulter le même document. Pour éviter cette laborieuse recherche, nécessaire tout de même pour l'obtention d'un document pour la première fois, certains outils peuvent être offerts. Nous avons vu dans la table 2.2 page 11, les outils classiques pour faciliter cette recherche. Certains de ces outils, comme les points de repère (par défaut ou définis par marquage des documents intéressants), permettent à l'utilisateur de retrouver un document préalablement parcouru. Dans la mesure où l'organisation et la présentation de ces repères sont bien faites (par une description claire des documents et par une vue d'ensemble compréhensible et facilement manipulable), l'utilisateur peut effectuer cette tâche plus rapidement.

Les interfaces 2.5D et 3D peuvent profiter de notre agilité naturelle pour localiser les informations dans l'espace. De nombreux travaux s'accordent sur l'importance de la mémoire spatiale pour retrouver dans des paysages informationnels des informations préalablement consultées [Darken96, Jones86, Ruddle97, Waller98]. Les systèmes de visualisation décrits dans la suite de ce rapport utilisent quasiment tous la cognition spatiale, une des plus importantes et incontournables possibilités offertes par les interfaces 3D. Ils se différencient par le type d'organisation des données. Certains laissent l'utilisateur évoluer dans une organisation spatiale automatique des données ; alors que d'autres laissent l'utilisateur positionner librement les données afin que la mémoire spatiale soit encore plus stimulée [Card91, Czerwinski99, Robertson98, Robertson00].

## 2.3 Les différents domaines d'application

Nous couvrons dans cette partie les différents domaines d'application des techniques de visualisation. Les bases de données et les processus ont été les deux domaines qui ont le plus bénéficié de ces nouvelles techniques. Mais la visualisation d'hyperstructures semble combler son retard depuis l'explosion d'Internet.

La majorité des systèmes que nous décrivons ici sont des prototypes ou des démonstrateurs. Cela implique que leurs caractéristiques, leurs points forts et leurs lacunes ne peuvent être évalués qu'à travers les publications associées. Par conséquent, cette section synthétise les publications décrivant ces systèmes plutôt que de fournir une critique objective.

### 2.3.1 Les réseaux

En 1993, a été développé chez *British Telecom*, un prototype utilisant des techniques 3D pour visualiser un réseau téléphonique [Walker93]. Cet environnement virtuel permet aux ingénieurs d'évoluer (de naviguer) à travers le réseau et de jouer sur le niveau de détail des informations. Ce système se rapproche des techniques utilisant le zoom et le filtrage (techniques que nous verrons dans la section 3.2.2 page 67) puisqu'il autorise à la fois une vue d'ensemble et une vue détaillée de certains objets. L'avantage d'un domaine d'application

de ce type par rapport à d'autres domaines est l'organisation spatiale claire et naturelle de la structure visualisée. En effet, dans le cas d'un réseau téléphonique, la structure est physiquement réelle et géographiquement distribuée. De telles structures fournissent donc une bonne base pour des visualisations car elles ont déjà une certaine forme concrète et tendent ainsi vers des abstractions moins vagues.

Le réseau est visualisé sous la forme d'un graphe affiché au dessus d'une carte du Royaume Uni. Par un tel procédé, la corrélation entre les centraux et les lignes téléphoniques et la géographie est forte. Les informations associées à chacune objet peuvent être affichées ou masquées et des données additionnelles peuvent être obtenues en s'approchant des nœuds du réseau. Les disfonctionnements sont accentués pour être repérables rapidement et l'utilisateur peut mener son investigation en "entrant" dans un nœud particulier qui pose problème. A l'intérieur d'un nœud, l'utilisateur est transporté dans une pièce offrant une variété d'informations concernant le statut du nœud. Ces informations, sous forme de graphiques ou de données brutes, sont plaquées sur les murs de la salle.

### 2.3.2 Les bases de données

Les bases de données forment naturellement l'endroit, avec le *World Wide Web*, où nous trouvons le plus de données. La différence avec les données issues d'Internet vient de leur plus grande homogénéité. En effet, elles doivent généralement satisfaire un certain nombre de critères pour être formatées et s'intégrer à une base, alors que sur Internet, il n'y a pas nécessairement de relations sémantiques entres les pages qui se référencent. Par conséquent, les bases de données bénéficient amplement des techniques d'organisation vues précédemment, ce qui facilite grandement leur présentation à l'utilisateur. Les projets suivants sont parmi les plus cités et les plus fondateurs dans le domaine de la visualisation des bases de données.

#### AMAZE

AMAZE [Benford94a, Boyle93, Boyle96] est un langage graphique tridimensionnel d'interrogation de bases de données créé spécifiquement pour l'analyse des données structurelles de protéines. Ce projet a été développé à la suite d'une série d'améliorations portant sur l'interface d'accès à la base de données existante. AMAZE permet aux utilisateurs de construire visuellement une requête et d'explorer les résultats obtenus dans un même environnement 3D. Pour bâtir sa recherche, l'utilisateur agit directement sur la représentation tridimensionnelle du schéma de la requête. La requête ainsi réalisée est ensuite transmise à la base de données distante et les données résultantes sont affichées sous la forme d'un ensemble d'objets 3D. L'utilisateur peut étudier plus en avant ces objets pour obtenir de nouvelles informations et les utiliser pour affiner sa recherche.

L'interface de AMAZE mélange des éléments 2D et 3D afin d'étendre plutôt que de remplacer l'interface 2D traditionnelle souvent plus appropriée. Ce mélange 2D/3D est obtenu en affichant les environnements 3D dans des fenêtres 2D séparées. Les fenêtres représentent ainsi les différentes sessions et pourront servir aussi bien à afficher une scène 3D qu'une interface 2D.

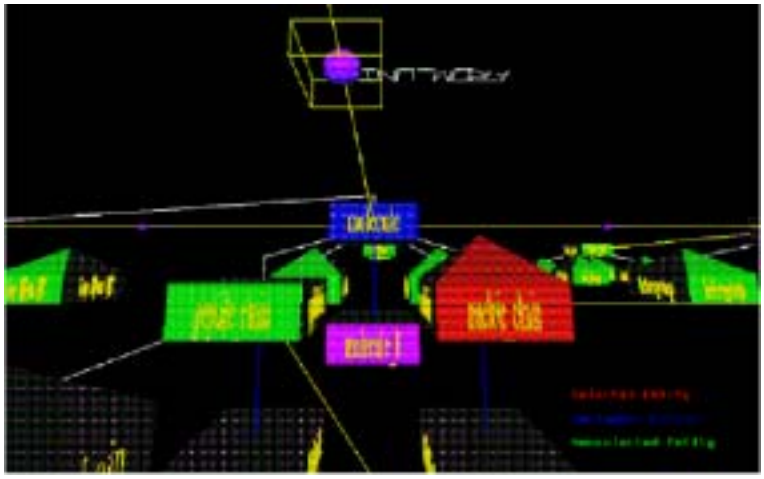

**Fig. 2.1** Un exemple de visualisation avec AMAZE [Boyle96].

Un utilisateur pratiquant l'interface d'AMAZE commence sa session en attachant des contraintes aux attributs appropriés par l'intermédiaire d'une zone de dialogue. Le résultat de la requête est construit en regroupant et en coloriant les objets de même type. Pour visualiser un objet particulier, l'utilisateur clique sur le cube qui le représente, ce qui à pour effet de lancer une visionneuse externe appropriée pour interpréter et visualiser les données.

## WINONA

WINONA[4] fournit une visualisation tridimensionnelle de la structure et du contenu d'une base de données objet. WINONA emploie principalement des variations du *perspective wall* (dans la section 3.2.3 page 80) et de l'arbre conique (dans la même section page 81) pour afficher la structure de la base de données. La représentation et la manipulation de ces structures 3D sont rendues possible par l'intermédiaire d'un système de fenêtrage 2D standard et d'une boîte à outils (*toolkit*) pour commander le point de vue. WINONA permet à l'utilisateur d'étudier et d'agir sur le contenu de la base de données à travers ces deux visualisations 3D qui montrent toutes les deux les mêmes informations d'une manière différente (l'une de manière séquentielle et l'autre de manière hiérarchique).

La visualisation hiérarchique de type *cone tree* montre les relations entre les classes de la base de données. Chacune des classes est représentée par une face perpendiculaire au plan de l'arbre. Dans chaque classe, les différents attributs sont représentés sous la forme de cubes. La taille, la couleur et le niveau de détail de ces cubes augmentent lorsque les attributs correspondants sont sélectionnés. Les relations entre les classes et les données sont montrées sous forme de liens (comme on peut le voir dans la figure 2.2). Ces relations sont accentuées si un des objets liés est sélectionné. Le *perspective wall* montre les mêmes informations sous une forme similaire, avec les objets affichés sur des sections du mur.

[4] WINdows Object Navigation Application.

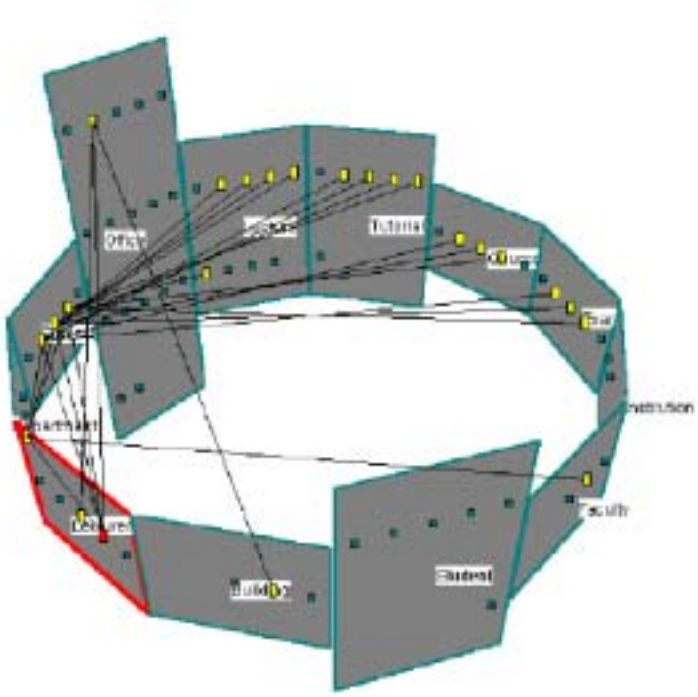

**Fig. 2.2** Un exemple de visualisation circulaire avec Winona.

### 2.3.3 Travail collaboratif

Il existe un certain nombre d'interfaces 3D pour le travail collaboratif, parmi lesquelles Spin [Dumas99a], qui n'ont pas pour but premier de représenter un espace de données. Dans le domaine de la visualisation d'informations, on parlera plutôt de PIT (*Populated Information Terrain* - paysages informationnels peuplés) que de travail collaboratif. Le concept d'un PIT est de fournir une visualisation efficace en s'appuyant sur les idées principales issues des domaines du travail coopératif assisté par ordinateur (TCAO), de la réalité virtuelle et des bases de données. Un PIT peut être défini comme un espace virtuel de données (paysage d'informations) qui est habité (peuplé) par des utilisateurs multiples [Benford94b, Colebourne94].

La philosophie guidant les PITs est donc de soutenir le travail coopératif *dans* les données par opposition à simplement *avec* les données. Les PITs peuvent convenir à n'importe quelle application disposant d'un ensemble de données sauvegardées. Il faut néanmoins que la collaboration sur ces données soit judicieuse et que l'on dispose d'abstractions visuelles appropriées. Les ensembles de données peuvent être, par exemple, des catalogues de bibliothèques, de documents hypertextes ou encore de grands systèmes logiciels. Ils se distinguent donc des visualisations de bases de données vues précédemment par leurs aspects multi-utilisateurs et multidisciplinaires.

Les PITs ont été développés en réponse au manque d'outils permettant à la fois le travail coopératif et la navigation parmi les informations. La réalité virtuelle est utilisée pour créer un environnement riche en informations dans lequel l'utilisateur peut se diriger et

explorer librement. Elle offre aussi à l'utilisateur la possibilité de se sentir immergé dans les données tout en utilisant pleinement ses qualités perceptives et intuitives. Le travail coopératif est, quant à lui, supporté dans les PITs en peuplant l'environnement virtuel avec des utilisateurs. Ceci permet aux utilisateurs du PITS de se rendre compte de la présence et des actions des autres utilisateurs et de communiquer avec eux, indépendamment de leur localisation physique. Cette conscience qu'il existe d'autres utilisateurs dans un PITs permet un vrai partage d'informations et de savoir-faire. Un utilisateur intéressé par un domaine particulier peut interroger d'autres utilisateurs qui naviguent dans le même ensemble de données ou demander sa direction pour accéder aux données appropriées. Et toute aide dans un paysage informationnel n'est pas superflue.

## Q-PIT

Q-PIT [Benford94b] est un prototype de PIT qui a pris comme parti d'utiliser l'organisation de Benedikt. Q-PIT s'appuie sur la bibliothèque de réalité virtuelle *World ToolKit* et est capable de traiter et d'afficher une base de données simple. Il permet l'affichage, le questionnement et la manipulation de la base de données dans un environnement tridimensionnel. Mais également, comme tout PIT, il partage cet environnement entre des utilisateurs multiples.

Dans Q-PIT, les attributs des données visualisées sont associés aux dimensions extrinsèques et intrinsèques de leurs représentations 3D. Les dimensions extrinsèques (celles portant une information métrique) utilisées sont les axes X, Y et Z du repère cartésien. Les dimensions intrinsèques (celles portant sur l'apparence des données) sont, quant à elles, la forme, la taille et la vitesse angulaire. Les utilisateurs sont personnifiés sous la forme de monolithes (c'est-à-dire comme des grands cubes minces et allongés) ayant chacun une couleur différente pour représenter leur identité. Chacun se rend ainsi compte de la présence d'autres utilisateurs et peut obtenir des informations ou communiquer avec ces autres utilisateurs.

La manipulation des données est rendue possible par l'intermédiaire d'un certain nombre de méthodes. L'utilisateur peut choisir une donnée élémentaire par sa représentation 3D ou construire une requête ayant pour résultat un certain nombre de données élémentaires pertinentes. Les données peuvent être modifiées en choisissant un objet puis en employant une interface textuelle pour modifier ses champs. Une fois les attributs modifiés, les changements liés aux dimensions intrinsèques ou extrinsèques sont immédiatement reflétés dans la visualisation, grâce à une transition fluide pour que les utilisateurs ne perdent pas de vue les données modifiées.

## BEAD

BEAD [Chalmers92] est un système de visualisation de documents qui s'appuie sur la boîte à outils et le visualiseur DIVE ([Carlsson93b, Carlsson93a]) pour produire un environnement de travail collaboratif. BEAD se sert d'une variante de la technique du recuit

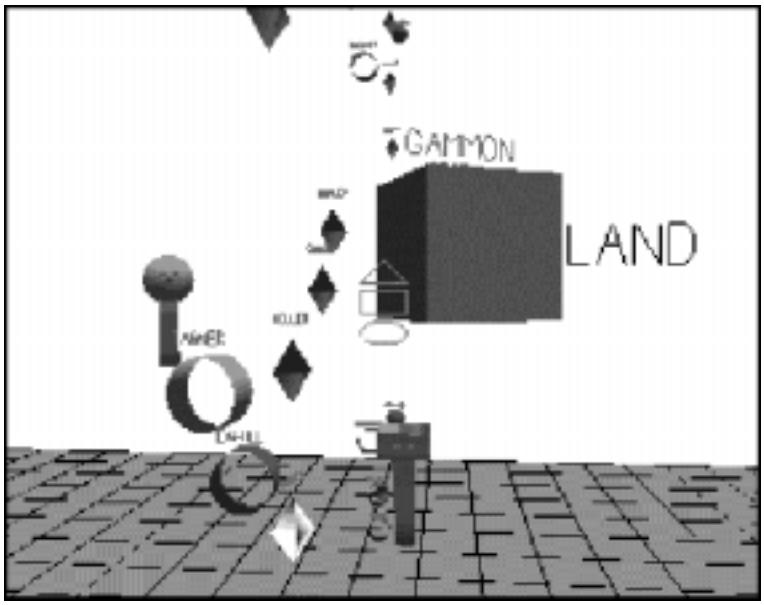


**Fig. 2.3** Un exemple de visualisation avec Q-PIT [Benford94b].

simulé[5] afin d'arranger les représentations de documents dans l'espace 3D et former un paysage d'informations. L'algorithme de disposition se base sur la similitude des documents pour produire un paysage dans lequel les documents étroitement liés ou semblables sont regroupés géographiquement. Pour ce faire, l'algorithme utilise les occurrences de mots dans les différents documents. Deux documents semblables (utilisant le même champ lexical) seront ainsi attirés l'un vers l'autre, tandis que deux documents différents se repousseront.

Dans BEAD, les distances entre les documents correspondent à une mesure de similitude. Pour cela, les documents se déplacent librement dans les trois dimensions afin de former une boule ou un nuage. Suite à cette implémentation, il a été constaté que la complexité visuelle de telles structures et les problèmes inhérents à la navigation rendaient l'interface inefficace. Le développement de BEAD s'est alors orienté vers un environnement utilisant essentiellement deux dimensions avec lesquelles les documents forment un paysage plus plat. Cependant, il est toujours possible d'associer à un document un certain degré de liberté vertical. Les documents sont alors rassemblés par des formes polygonales produisant une visualisation ressemblant à une île dont le paysage ondule. La figure 2.4 montre le paysage construit par BEAD avec 500 références bibliographiques.

L'environnement virtuel produit par BEAD peut être peuplé par de multiples utilisateurs (sans quoi il ne ferait pas partie des PITs). Ils y sont représentés par des géométries en forme de `T` sur lesquelles sont dessinés deux yeux. Ceci permet de voir exactement comment

[5]Nom d'un algorithme classique permettant d'obtenir rapidement une valeur approchée d'une solution d'un problème NP-complet (e.g. problème du voyageur de commerce). L'idée est que si on a un problème contenant beaucoup de conditions et qu'on a une idée de la meilleure solution, on prend une solution au hasard, puis on teste toutes les conditions. À chaque fois qu'une condition n'est pas respectée, on modifie légèrement la solution pour qu'elle la respecte, puis on re-teste toutes les conditions jusqu'à ce que ce soit bon (ou qu'on ait dépassé le temps imparti). L'expérience montre qu'on tend ainsi très rapidement vers un minimum local proche de la meilleure solution.

un utilisateur est orienté et dans quelle direction il regarde. En naviguant dans le paysage, certaines caractéristiques comme les secteurs denses ou un terrain accidenté peuvent apparaître de manière évidente. Ces caractéristiques fournissent des informations additionnelles sur la similitude entre documents. L'algorithme de disposition a une tendance à pousser vers les rivages de l'"île" tous les documents qui sont étrangers au thème de la recherche. Les caractéristiques géographiques telles qu'un rivage, un groupement dense ou un secteur accidenté permettent aux utilisateurs d'apprendre rapidement et d'accumuler l'expérience nécessaire pour se diriger parmi les données et trouver celles qui répondent à leurs besoins.

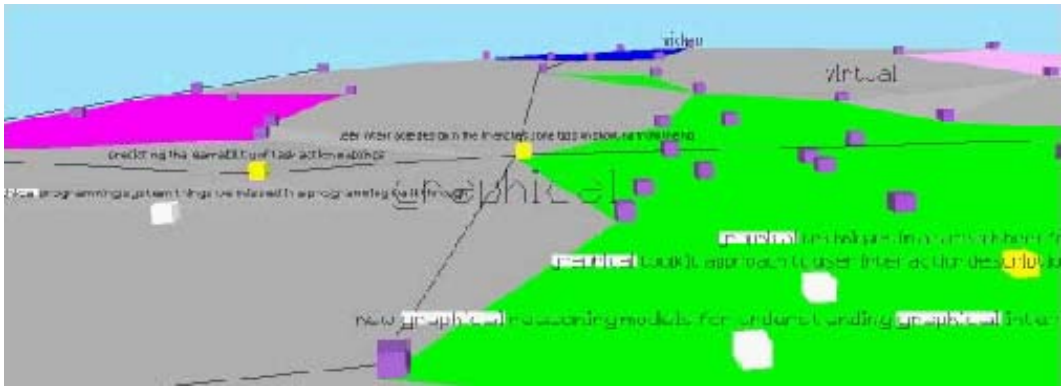

**Fig. 2.4** Un exemple de visualisation avec BEAD [Chalmers92].

## VIBE et VR-VIBE

Le système originel dont VR-VIBE est issu, VIBE [Olsen93] (dont on peut voir une copie d'écran dans la figure 2.5), a été développé à l'université de Pittsburgh. Il fournit aux utilisateurs une visualisation 2D d'une collection de documents répondant à la requête des utilisateurs. Ceci est rendu possible dans VIBE grâce à la construction d'un ensemble de points d'intérêt (PDI) à partir des mots-clés qui sont employés dans les requêtes. La visualisation est construite en distribuant les points d'intérêt sur le plan 2D puis en plaçant les icônes des documents entre les PDIs en tenant compte de leur pertinence avec chaque PDI. Par exemple, avec deux PDIs A et B, un document peut avoir une pertinence de 4 au PDI A et une pertinence de 3 au PDI B. Ce document sera alors placé sur le segment AB à une distance de A égale à 3/7 de la longueur de AB. Avec plusieurs PDIs, il suffit de calculer le barycentre en pondérant chaque PDI avec le taux de pertinence du document.

VR-VIBE utilise des techniques 3D pour tenter de régler les problèmes liés à l'affichage de nombreux documents et pour permettre également l'emploi d'un plus grand nombre de PDIs. Avec VR-VIBE, la technique pour déterminer la position d'un document en fonction des PDIs est la même que dans VIBE sauf qu'elle est étendue aux trois dimensions. Pour la politique de placement des PDIs et des documents, VR-VIBE offre deux méthodes légèrement différentes. La première est très semblable à la politique de placement de VIBE puisqu'elle contraint les objets à rester dans un plan 2D. Un déplacement vertical est ensuite appliqué à chaque objet pour représenter leur degré d'importance.

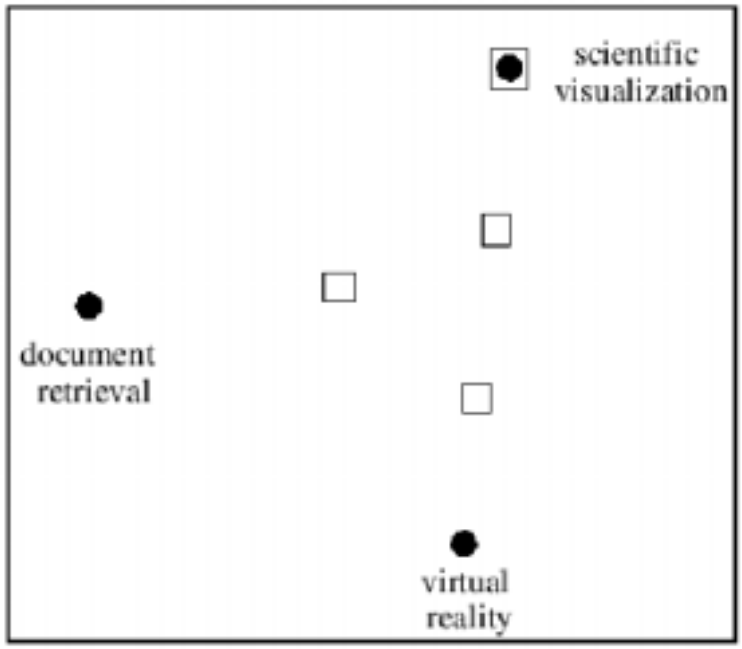


**Fig. 2.5** Un exemple de visualisation en 2D avec VIBE [Olsen93].

Dans la deuxième méthode (illustrée dans la figure 2.6 qui montre une copie d'écran d'une session VR-VIBE avec 5 PDIs), les objets peuvent être placés en un point quelconque de l'espace 3D. Les dimensions intrinsèques des objets telles que leur couleur sont exploitées pour indiquer la sélection et pour souligner leur pertinence, leur niveau d'importance par rapport aux PDIs. Ceci permet de faire la distinction entre deux documents qui ont tous deux les mêmes proportions de pertinence. Par exemple, un document ayant une pertinence de 3 et 3 avec deux PDIs serait colorié différemment d'un document ayant les pertinences 8 et 8. La forme est également employée pour distinguer les documents, les PDIs et les utilisateurs respectivement représentés comme des cubes, des octaèdres et des monolithes (comme dans Q-PIT).

VR-VIBE autorise les utilisateurs à se déplacer librement dans le paysage des informations, à choisir des documents, à exécuter des requêtes, à appliquer des filtrages ou à demander des informations additionnelles. L'interface 3D permet un certain nombre d'interactions comme ajouter des PDIs, activer ou désactiver les PDIs ou déplacer les PDIs pour voir quels documents sont attirés avec eux. Les utilisateurs peuvent également définir un certain nombre de points de vue dans la visualisation et commuter de l'un à l'autre à volonté. Ceci permet aux utilisateurs de comparer différentes vues mais c'est également une aide à la navigation puisque ces points de vue sont des points de référence auxquels il est facile de retourner rapidement. Enfin, VR-VIBE est un environnement multi-utilisateurs dans lequel tout utilisateur est conscient de la présence des autres et peut profiter de ce partage de l'environnement pour se guider ou obtenir des informations.

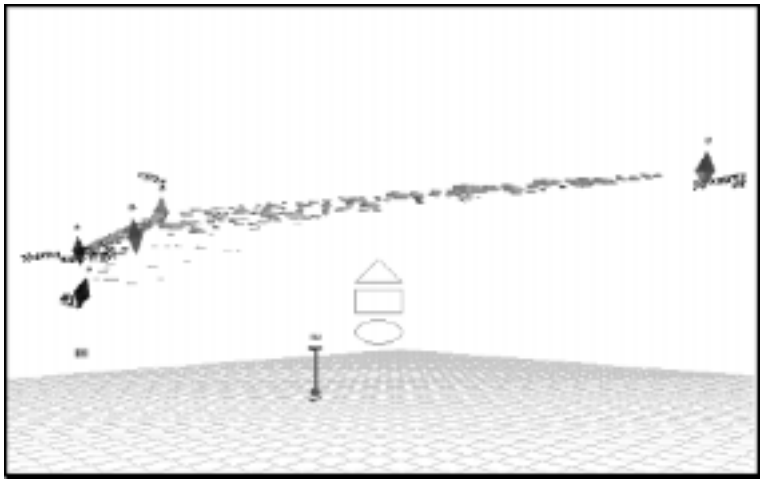

**Fig. 2.6** Un exemple de visualisation en 2D avec VR-VIBE.

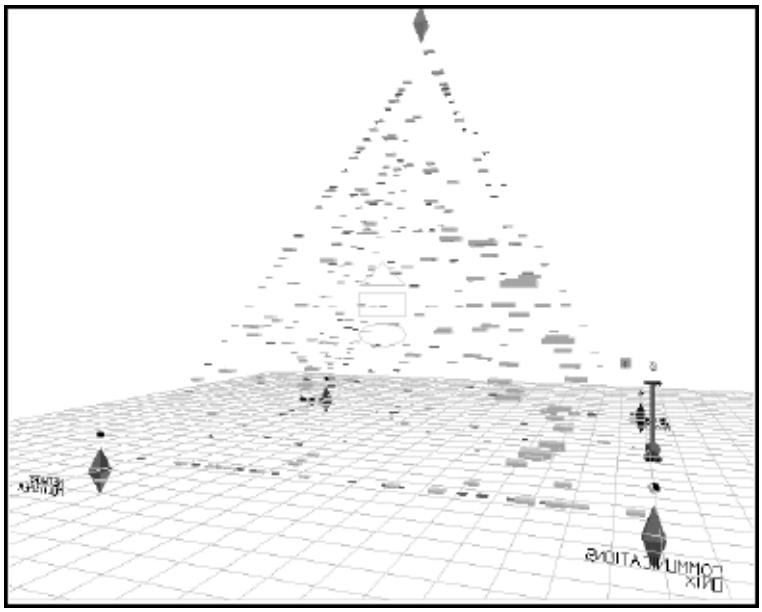

**Fig. 2.7** Un exemple de visualisation en 3D avec VR-VIBE.

## LyberWorld

LyberWorld [Hemmje93, Hemmje94] est un prototype d'interface utilisateur pour la recherche documentaire qui vise à utiliser pleinement les avantages fournis par l'infographie 3D. LyberWorld se concentre sur la visualisation d'un espace abstrait d'informations et complète le système d'interrogation INQUERY [Callan92]. Les visualisations utilisées dans LyberWorld fournissent une interface intuitive pour rendre la recherche efficace et la lecture rapide. Le but de LyberWorld est de permettre une corrélation forte entre le modèle graphique des paysages informationnels et le modèle spatial mémorisé par l'utilisateur. Si

une telle corrélation existe, les visualisations devraient alors aider l'utilisateur à explorer et à interroger la base de données, à se diriger et à s'orienter dans les données, et à juger de la pertinence et du contexte de la réponse à une requête.

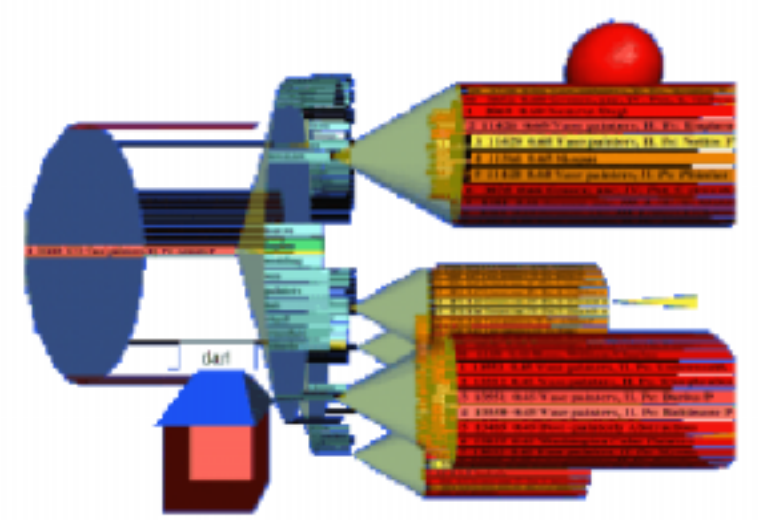

**Fig. 2.8** Les cônes de navigation de LyberWorld [Hemmje94].

LyberWorld modèle l'espace sous la forme d'un réseau de documents et de termes. Dans ce réseau, les relations sont constituées à partir d'une quantification de la pertinence entre les documents et les termes. Deux visualisations différentes sont principalement utilisées dans LyberWorld, les cônes de navigation (*NavigationCones* que l'on peut voir dans la figure 2.8) et la sphère de pertinence (*RelevanceSphere* dans la figure 2.9). Les premiers fournissent une visualisation de l'historique de l'exploration, c'est-à-dire jusqu'à quel point l'espace des données a été exploré et, également, les requêtes qui ont permis de se frayer un chemin parmi les données. La sphère de pertinence, quant à elle, fournit un affichage des documents recherchés et de leur importance par rapport à chacun des termes de recherche employés dans la requête. De plus, la *RelevanceSphere* fournit une indication visuelle sur les groupes de documents en faisant ressortir les documents étroitement liés.

Comme on peut le constater dans la figure 2.8, les *NavigationCones* s'inspirent des arbres coniques [Robertson91] (section 3.2.3 page 81) pour représenter une structure extensible et rétractable des régions de l'espace explorées. Une limitation due à ces arbres coniques est l'impossibilité de visualiser des structures de données non hiérarchiques. Or, un réseau complexe de documents est souvent loin de se présenter sous cette forme. LyberWorld surmonte ce problème en transformant le réseau de documents-termes en une structure hiérarchique grâce à un mécanisme de redondance. Le réseau est ainsi transformé en un arbre dont les nœuds sont alternativement des documents et des termes sur chaque rangée. Ceci est effectué en dupliquant des nœuds selon les besoins et en introduisant de nouveaux liens entre les nœuds dupliqués.

La navigation dans ce paysage informationnel est faite en dévoilant des nœuds d'intérêt. Ces nœuds sont colorés de manière à indiquer leur type (document ou terme). Les nœuds dupliqués sont accentués avec une couleur et une forme différente pour mettre en valeur un chemin caché. Les nœuds d'une rangée particulière de l'arbre sont ordonnés en fonction de leur taux de pertinence avec leur nœud parent. Comme avec les arbres conique de Robertson *et al.* [Robertson91], les *NavigationCones* tournent sans à-coup pour positionner le nœud d'intérêt courant le plus proche du point de vue. De même, lorsque l'arbre est déployé ou rétracté, les rayons des diverses rangées sont ajustés dynamiquement pour maintenir une structure compacte tout en évitant les collisions.

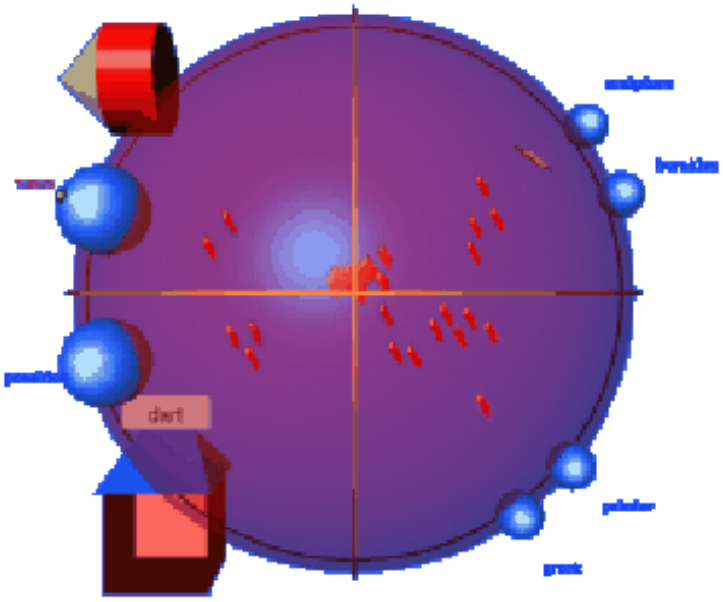

**Fig. 2.9** La sphère de pertinence de LyberWorld [Hemmje94].

La sphère de pertinence facilite l'appréhension de la pertinence des éléments présents dans l'espace de résultat. La *RelevanceSphere* tente de régler le problème d'ambiguïté présent dans les visualisations VIBE et VR-VIBE vues précédemment. Cette ambiguïté se produit parce qu'à un point particulier dans l'espace, peut être attribué un certain nombre de taux de pertinence par rapport à différents points d'intérêt (PDIs). Ce problème est décrit par Hemmje *et al.* [Hemmje94] et également par Benford *et al.* [Benford95]. La *RelevanceSphere* règle ce problème en transformant le réseau en une représentation spatiale dans laquelle les taux de pertinence sont visualisés comme des distances qui respectent toujours les proportions numériques. Cette approche permet de comparer facilement les distances et donc les taux de pertinences.

La sphère de pertinence, tout comme VIBE et VR-VIBE, est le résultat de calculs basés sur des vecteurs d'attraction et des chemins de pertinence. Dans la *RelevanceSphere*, tous les nœuds termes sont placés sur la surface d'une sphère et tous les nœuds documents sont placés à l'intérieur de la sphère en fonction de leur pertinence à chacun des termes. Du point

de vue pratique, la position d'un nœud document est calculée en additionnant les vecteurs d'attraction définis entre le nœud et chacun des termes. Le vecteur résultant déterminera sa position dans la sphère, en utilisant le centre de la sphère comme origine. Cette visualisation permet à l'utilisateur non seulement de voir la pertinence des documents par rapport à la requête globale mais également leur pertinence relative par rapport à chacun des termes. Pour la pertinence globale, plus le document sera proche de la surface de la sphère, plus il sera pertinent. La pertinence avec chaque terme, quant à elle, est dénotée par la distance séparant le document du terme.

L'utilisation de l'espace tridimensionnel pour la sphère de pertinence offre des avantages mais aussi des inconvénients. Le volume d'affichage supplémentaire (et par conséquent la liberté de positionnement accrue) réduit la probabilité de positionnement peu clair des nœuds, bien qu'il soit tout de même possible qu'une ambiguïté soit présente. Un problème souvent rencontré dans beaucoup d'interfaces 3D est la trop grande confiance que l'on peut porter sur les capacités des utilisateurs à percevoir la profondeur. Lorsque l'on regarde une image 2D statique d'une structure 3D, il est souvent difficile d'appréhender la profondeur, la taille et le positionnement des éléments. La *RelevanceSphere* traite ce problème en permettant à l'utilisateur de tourner la sphère librement pour l'examiner de n'importe quel angle (comme s'il examinait une pomme dans sa main). Des interactions supplémentaires permettant à l'utilisateur d'agir sur la sphère sont fournies pour étudier et manipuler l'espace de résultat. L'utilisateur peut, par exemple, changer des paramètres tels que la densité des documents, l'attraction exercée par les termes ou encore l'échelle de la visualisation. Ces interactions sont importantes pour augmenter la compréhension qu'à l'utilisateur des résultats d'une requête et de la structure de visualisation.

### Vineta

Vineta est un prototype de système de visualisation d'informations développé par U. Krohn [Krohn95, Krohn96a, Krohn96b]. De la même manière que VR-VIBE, LyberWorld et BEAD, Vineta présente des documents et des termes comme des objets graphiques éparpillés dans un espace tridimensionnel, l'espace de navigation. Le positionnement de ces objets dans cet espace est directement lié à la pertinence sémantique entre les documents, les termes et les intérêts de l'utilisateur. Vineta diffère des autres visualisations dans le nombre de dimensions ou de termes qu'il peut gérer. Des systèmes tels que VR-VIBE sont limités par le nombre de termes qu'ils peuvent présenter simultanément. Vineta applique des techniques d'analyse multi-variable et d'algèbre linéaire numérique pour déterminer la position des documents et des termes dans l'espace tridimensionnel de navigation. Cette réduction d'un système $nD$ vers un système d'affichage 3D permet de visualiser des ensembles de termes beaucoup plus grands.

Krohn déclare que la navigation informationnelle peut être fortement liée à la compréhension intuitive des faits abstraits grâce aux analogies que l'on peut faire avec des concepts familiers tels que la localisation ou le mouvement. Vineta emploie la proximité spatiale pour représenter la similitude sémantique entre les objets (documents). Comme avec les systèmes précédemment décrits, les documents semblables sont positionnés dans une même région de l'espace tandis que les documents différents sont éloignés.

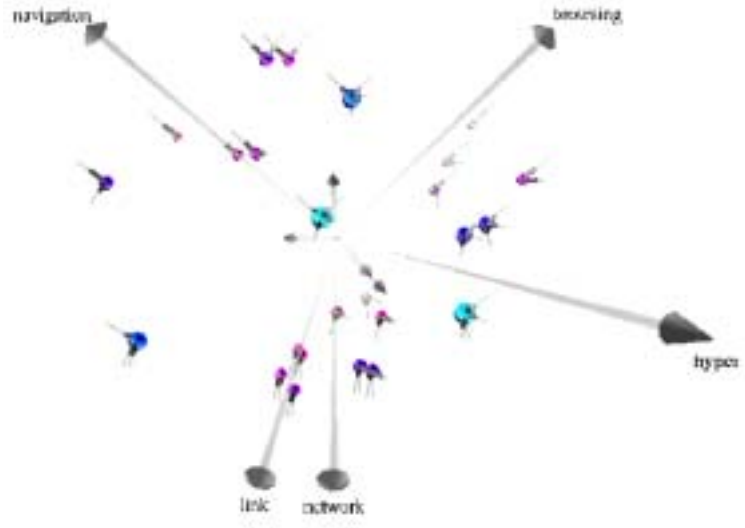


**Fig. 2.10** La métaphore de galaxie de Vineta [Krohn96b].

Vineta emploit deux métaphores principales pour produire ses visualisations, la métaphore de la galaxie et la métaphore du paysage (*landscape*). La dernière a remplacé la première dans la version finale de Vineta après avoir prouvé qu'elle était plus intuitive et plus facile à comprendre.

La métaphore de galaxie représente l'espace de navigation comme une galaxie d'étoiles dans laquelle les documents sont des étoiles fixes tandis que les termes de recherche sont représentés comme des étoiles filantes (plus comme des axes). La similitude sémantique est dénotée par la proximité des étoiles. La figure 2.10 est une copie d'écran de Vineta montrant la métaphore de la galaxie. Les termes y sont représentés sous la forme de flèches tandis que les documents sont représentés par des sphères.

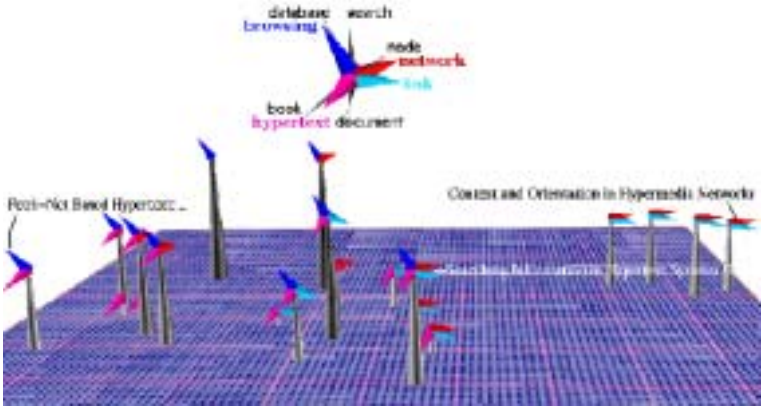


**Fig. 2.11** La métaphoren du paysage de Vineta [Krohn96b].

La métaphore du paysage représente l'espace de navigation comme une surface plate et texturée, plantée de fleurs avec des tiges et des pétales. La surface texturée du paysage aide le visionneur à appréhender la profondeur ou la distance des documents (fleurs). Les fleurs les plus proches du point de vue de l'utilisateur sont celles ayant le plus d'importance par rapport à la question initialement posée. Les tiges des fleurs aident l'utilisateur à extrapoler la position de la fleur sur le plan texturé servant de sol. Les pétales sur les fleurs représentent les termes de recherche et leur direction et leur couleur dénotent leur pertinence avec chacun de ces termes. La figure 2.11 est une copie d'écran montrant la métaphore du paysage. Nous pouvons voir dans cette image que la présence du sol aide considérablement pour déterminer les positions relatives des documents.

## VR-Mapper

L'outil VR-Mapper [Benford93] provient d'un système de communication appelé CyCo qui est basé sur une métaphore de salles (voir les salles 2D page 65 et les salles 3D page 83). Bien que cette métaphore soit un bon système pour la communication, la navigation dans un espace de conférence n'est pas simple. VR-Mapper a été développé pour régler ces problèmes en fournissant une représentation visuelle de l'espace de communication et en permettant également d'obtenir des informations sur l'environnement.

Un fichier de données, dans lequel sont énumérés toutes les salles et tous les chemins pour passer d'une salle à l'autre, décrit complètement chaque environnement de communication. La première version de VR-Mapper prenait ce fichier de données en entrée et montrait les informations sous forme d'un arbre dessiné dans une fenêtre classique. Cette représentation 2D était tout à fait suffisante lorsque de petits fichiers de données devaient être visualisés. Mais, pour les plus grands espaces de communication, la visualisation devenait très rapidement compliquée et confuse. Pour tenter d'obtenir une meilleure visualisation de ces grands espaces, VR-Mapper a subi une première modification afin de présenter les salles sous forme de cercles concentriques (figure 2.12). Cela a amené une légère amélioration mais ne permettait toujours pas d'afficher de nombreuses salles. La deuxième amélioration de VR-Mapper a donc été de lui accoler une interface 3D qui est non seulement une solution pour afficher de nombreuses données, mais aussi, plus généralement, pour représenter toutes sortes de données et non uniquement les forums de conférence pour lesquels ce projet avait été initialement développé.

VR-Mapper est ainsi devenu un environnement de visualisation très complet, capable de visualiser de nombreuses structures de données hiérarchiques en utilisant des représentations différentes. Trois représentations sont accessibles par trois modules que l'utilisateur peut sélectionner en fonction de ses préférences ou/et de la complexité de l'ensemble de données. Le premier module, celui de la figure 2.12, permet de visualiser les structures arborescentes en 2D de manière classique. Le second est une variation du premier (également dans la figure 2.12) dans lequel les nœuds sont positionnés sur des cercles concentriques entourant leur nœud parent. Le troisième module utilise les avantages de la troisième dimension en disposant les nœuds dans un arbre conique. Cette disposition est illustrée par la figure 2.13 dans laquelle est représentée une structure arborescente d'un système de fichiers UNIX et la consultation d'un fichier dans une fenêtre standard.

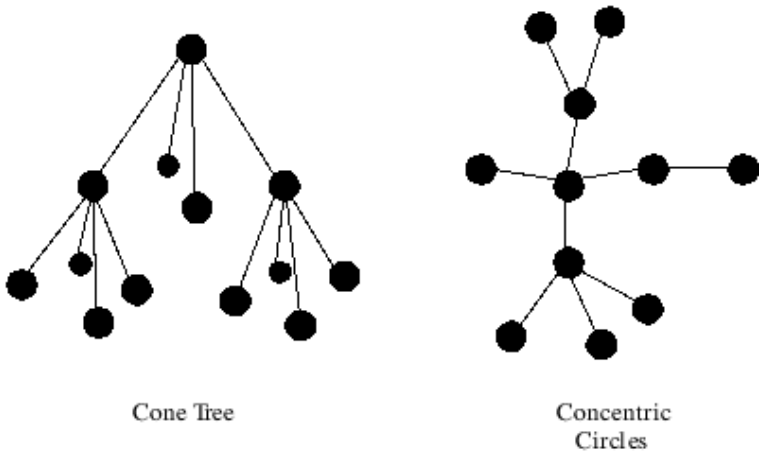


**Fig. 2.12** Les visualisations 2D de VR-Mapper [Benford93].

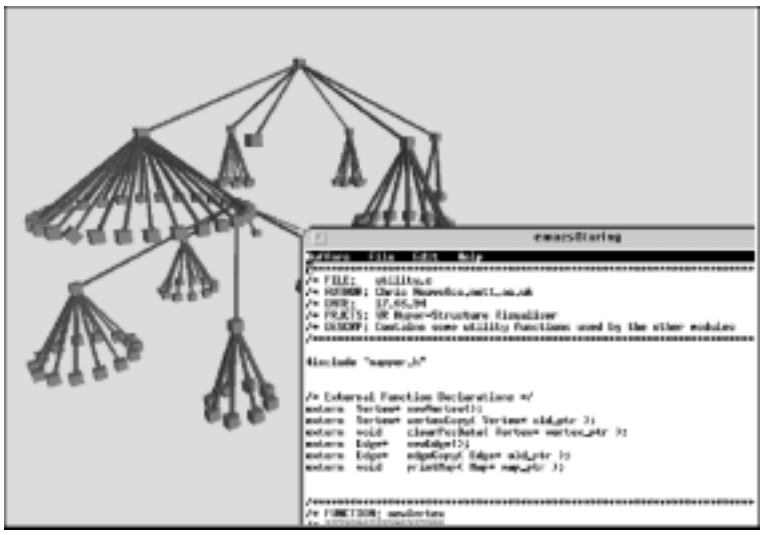

**Fig. 2.13** La visualisation 3D de type *cone tree* de VR-Mapper [Benford93].

### 2.3.4 Les hypertextes

Les hyperstructures sont constituées en général par un grand nombre d'informations liées entre elles par un ensemble de relations dont on ne connaît pas *a priori* la configuration. La complexité et l'organisation arbitraire de telles structures en font des ensembles très difficile à visualiser. Des hyperstructures relativement simples peuvent rapidement encombrer une visualisation et par conséquent détériorer son efficacité. Il est par conséquent nécessaire de disposer d'outils de visualisation adaptés. Un exemple d'hyperstructure est donné dans la figure 2.14.

Plusieurs systèmes ont été développés pour visualiser ces structures complexes. La majorité d'entre eux emploient des techniques de disposition dynamique de graphes (comme celles

**Fig. 2.14** Exemple d'hyperstructure.

décrites dans la section 3.2.1 page 66). Un nombre restreint de ces nombreux systèmes sont décrits ici. La plupart d'entre eux sont orientés vers la visualisation d'hypertextes qui sont les hyperstructures les plus couramment rencontrées. Les exemples suivants sont de ce fait plus particulièrement des techniques originales pour visualiser des structures issues du *World Wide Web*.

## Hyperspace

Hyperspace [Wood95] est issu du système Narcissus [Hendley95b] (voir page 45) initialement développé pour la visualisation de composants systèmes ou logiciels. Cependant, puisque Narcissus permet plus largement d'afficher des hyperstructures, une application toute trouvée est la présentation de structures hypermédias. Hyperspace a donc permis à la fois de poursuivre l'étude du système Narcissus mais également de s'attacher à la visualisation de fonds documentaires hypertextes. Pour la collecte de données, le système utilise le navigateur Mosaic du NCSA. Ce navigateur est également employé pour afficher dans la visualisation les pages HTML choisies.

Dans Hyperspace (et dans Narcissus également) la visualisation présente des graphes 3D en "fil de fer" dont les pages sont représentées comme des nœuds sphériques et les liens comme des flèches. La disposition du graphe est opérée en utilisant une technique d'auto-organisation (graphes dynamiques), avec insertion si nécessaire de nouveaux nœuds pour

clarifier la présentation. Le problème majeur de cette technique est que la forme de la structure, et donc l'aspect du modèle produit, peuvent changer considérablement quand un nouveau nœud ou un nouveau lien est inséré. La visualisation d'une même structure peut ainsi être différente d'une exécution à l'autre. Le modèle cognitif qu'a l'utilisateur de la structure devient par conséquent inefficace et, lors de l'exécution, il a besoin de se réorienter dans le graphe à chaque mise à jour.

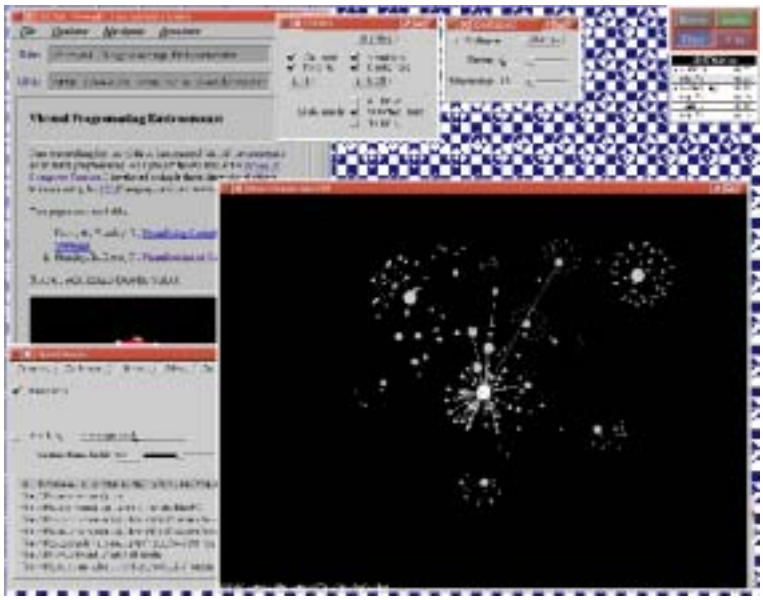

**Fig. 2.15** Hyperspace : Narcissus utilisé pour visualiser le Web [Wood95].

## WWW3D

WWW3D [Snowdon96] peut être comparé au système VR-Mapper [Benford93] car il visualise le *web* comme un graphe, composé d'une collection de nœuds (de pages) et de liens (de la même manière que VR-Mapper visualise le système de fichiers UNIX). Un exemple de structure hypertexte visualisée avec WWW3D est donné dans la figure 2.16.

La structure visualisée se développe au fur et à mesure que les utilisateurs (puisqu'il s'agit d'un système multi-utilisateurs) visitent des pages. Ainsi, plutôt que de visualiser l'intégralité des liens lorsqu'un nœud est visité, les informations apparaissent dans la visualisation au gré des actions des utilisateurs. La visualisation peut de ce fait être vue comme un historique ou une carte d'exploration des pages *Web* visitées.

Pour minimiser les affichages, WWW3D se sert de niveaux de détails (LODs - *Level Of Details*). Vu de l'extérieur, un nœud est représenté par une sphère opaque et son contenu n'est pas affiché. Lorsqu'un utilisateur est à l'intérieur d'un nœud (quand la page correspondante est visitée), le contenu est affiché de la manière décrite plus tard. Les liens portent également un certain nombre d'informations utiles.

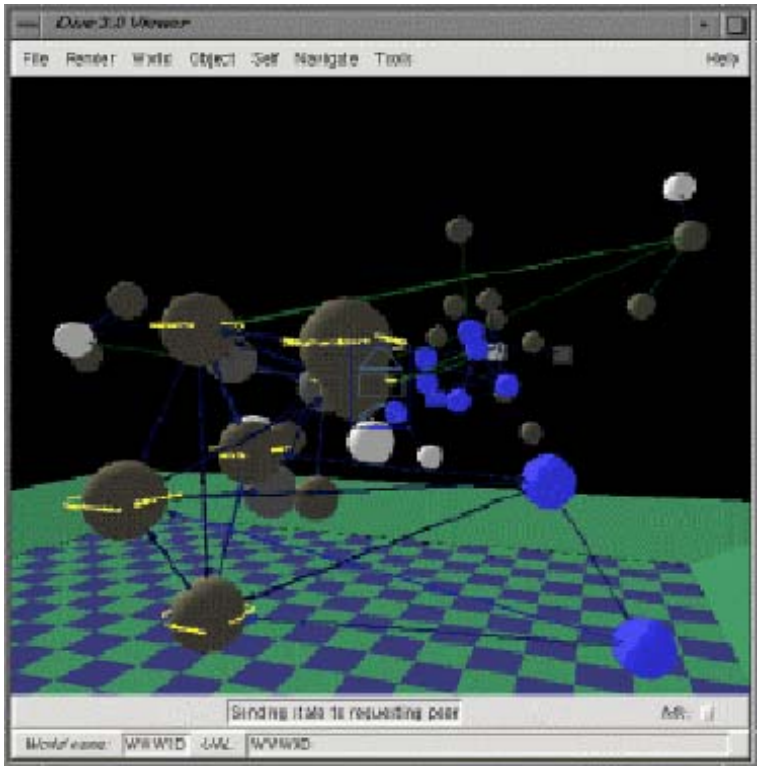


**Fig. 2.16** Une collection de pages HTML avec leurs liens avec WWW3D [Snowdon96].

Les caractéristiques visuelles des liens ont les significations suivantes :
- S'ils sont bleus, les documents liés sont sur le même serveur HTTP.
- Si les liens sont verts, les documents sont distants.
- La luminosité des couleurs dénote la durée depuis laquelle les liens n'ont pas été empruntés.
- Enfin, si WWW3D ne peut pas récupérer un document, les liens le référençant seront rouges.

Lorsqu'un utilisateur "entre" dans un nœud, il obtient immédiatement une schématisation 3D de la page correspondante (son contenu est plaqué sur la surface intérieure de la sphère – figure 2.17). Un texte y est représenté comme une icône dont le label est formé par les premiers du texte. Les images sont utilisées comme des textures et sont plaquées sur la sphère. Les liens vers d'autres documents sont symbolisés par des icônes dont le label associé indique la destination (l'URL). Cette manière d'intégrer les éléments d'une page HTML à l'intérieur d'une sphère n'est peut-être pas la plus lisible. Cependant, cela permet de garder une certaine homogénéité entre la visualisation de l'hyperstructure et la visualisation des documents (par l'utilisation des sphères).

## VizNet

VizNet est un système, développé par Fairchild *et al.* [Fairchild93], qui permet l'affichage de collections d'objets multimédias. Ce système sachant gérer les hyperstructures, est par

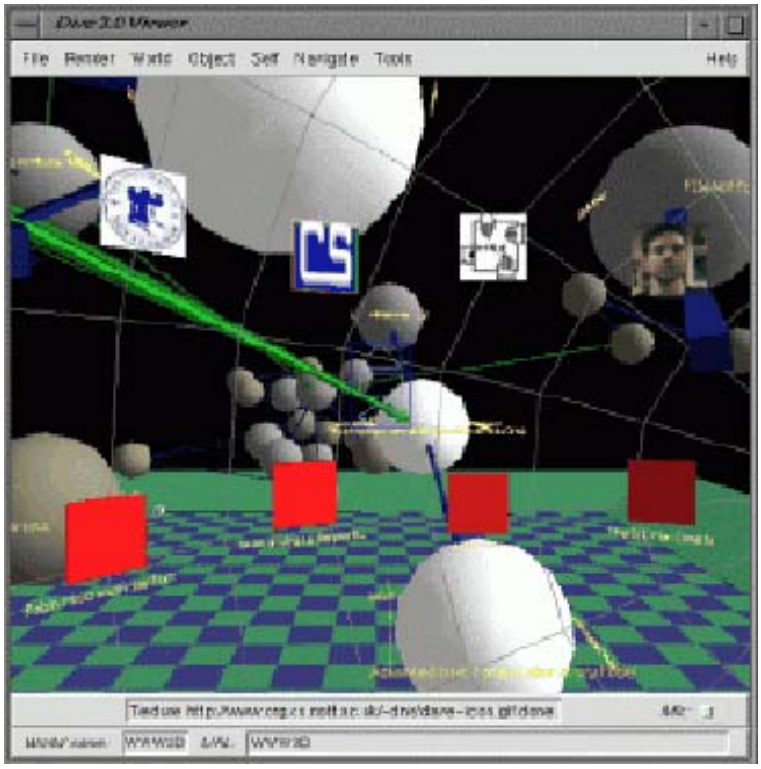

**Fig. 2.17** La représentation d'une page HTML avec WWW3D [Snowdon96].

conséquent naturellement spécialisé pour les documents hypertextes et hypermédias. Deux visualisations différentes sont produites par VizNet : un arbre conique (présenté dans la section 3.2.3 page 81) et une visualisation sphérique (présentée dans la même section page 82). Dans VizNet, ces deux techniques sont utilisées conjointement avec des vues de type *FishEye* pour afficher les objets multimédias. De plus, des niveaux de détail ou des filtres sur le contexte de l'objet sélectionné sont également implémentés. La taille et la résolution (ou la complexité géométrique dans le cas d'objets 3D) sont modifiées pour signifier le niveau d'intérêt ou d'importance d'un objet particulier.

Les objets graphiques sont attachés à des icônes et ces associations sont produites automatiquement en utilisant la technique *AutoIcon* également décrite dans [Fairchild93]. Dans le système VizNet, les objets tridimensionnels gardent leurs formes 3D, les images sont tracées sur des faces 3D, et les textes sont représentés dans l'espace par des textes 3D. Le niveau de détail choisi par l'utilisateur affecte chaque type d'objet différemment. Par exemple, dans le cas d'un objet 3D, c'est la complexité (le nombre de polygones) de l'objet qui sera changée. Pour une image, ce sera la résolution de la texture.

La figure 3.14 page 83 montre un exemple de visualisation produite par VizNet. Plus exactement, cette figure illustre la visualisation sphérique dans laquelle l'objet d'intérêt est affiché au centre et les objets relatifs sont éparpillés autour de lui.

## SemNet

SemNet [Fairchild88] a pour but premier d'expérimenter l'application des techniques 3D à la visualisation de grandes bases de connaissances (dont la plus connue est sans conteste l'Internet), en vue d'améliorer leur compréhension. Fairchild *et al.* ont pris pour hypothèses, que pour pouvoir travailler sur ces bases, un utilisateur doit avant tout comprendre :
– la signification des différents éléments,
– la position relative des éléments dans leur contexte hiérarchique,
– les relations explicites entre les éléments.
Les bases de connaissances représentent des informations sur les relations entre les objets. Or, Fairchild, Poltrock et Furnas pensent que les représentations graphiques fournissent une manière efficace de communiquer de telles relations. Le traitement visuel et l'utilisation de la mémoire spatiale permis par les systèmes 3D, en font de bons candidats.

Cependant, l'utilisation de la 3D ne peut pas, à elle seule, rendre une visualisation moins confuse et plus efficace. Pour simplifier l'interface, SemNet permet donc de réduire la complexité de la représentation visuelle de la base de données en masquant certaines informations. Par ailleurs, la disposition des éléments dans la visualisation peut considérablement accroître ou desservir, selon les cas, la compréhension de la structure de la base de connaissances. Dans SemNet, deux méthodes de disposition peuvent être utilisées. La première suppose la connaissance de relations sémantiques entre objets. Un algorithme dispose alors les objets en fonction de leur proximité sémantique. La deuxième méthode utilisée lorsque aucune information sémantique n'est donnée, ne prend en considération que les liens entre objets de telle manière que les objets fortement connectés soient proches.

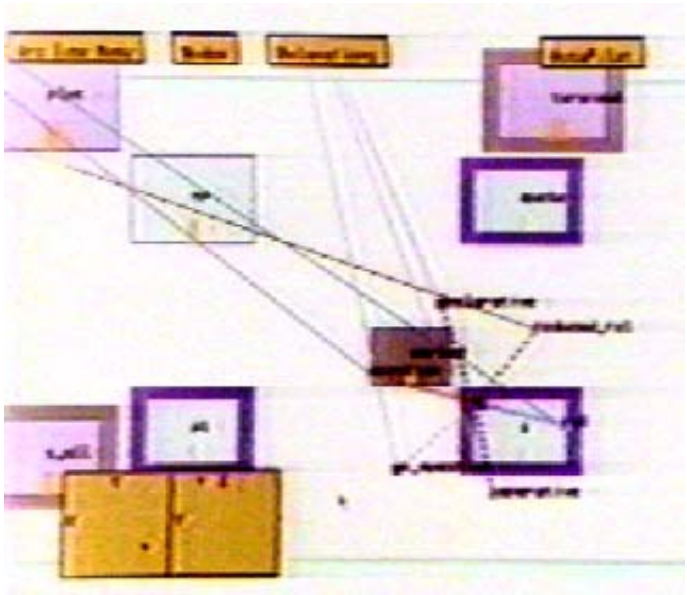

**Fig. 2.18** La visualisation SemNet [Fairchild88].

Le masquage des informations dans SemNet est effectué en utilisant deux techniques. La première est une simple réduction de détails en fonction de la distance séparant les objets du point de vue. De cette manière, les objets pas ou peu connectés aux objets vers lesquels l'utilisateur dirige son regard (son point de vue) sont automatiquement réduits en taille (par la perspective inhérente à la 3D) et en détails. On peut donc considérer qu'il s'agit d'une vue de poisson (*fisheye view*) basée purement sur la perspective tridimensionnelle. Pour obtenir également une réduction du contexte (l'ensemble de la hiérarchie des objets), la complexité du graphe est réduite tout en maintenant toujours le contexte global. Une hiérarchie de groupes est formée et seuls les objets des groupes d'intérêt sont affichés ; les autres groupes n'étant pas développés. Cette technique est similaire aux graphes nichés de Harel [Harel88] mais ressemble aussi aux géométries englobantes utilisées en 3D pour accélérer les tests de visibilité et de *clipping* [Foley90, Watt93].

## Web Forager

Le *Web Forager* [Card96] est une interface conçue pour organiser une collection de pages HTML. Cet outil de *data mining* (de collecte et d'oganisation des informations) propose une interface très conviviale et intuitive dont une capture d'écran est proposée dans la figure 2.19.

**Fig. 2.19** Le *Web Forager* [Card96].

Dans cette interface, l'organisation des informations se fait sur deux niveaux différents. Le premier autorise l'utilisateur à regrouper des pages HTML dans une sorte de classeur, les *Web Books*. Ils ont l'apparence et la structure séquentielle des livres comme on peut le voir dans la figure 2.20. L'utilisateur peut constituer autant de *Web Books* qu'il a de sujets d'intérêt. Chacun de ces "livres" renferme ainsi un certain nombre de pages HTML, de la même manière qu'un dossier contient un certain nombre d'URL dans une gestion classique par signets. Plusieurs outils sont offerts pour pouvoir manipuler ces classeurs,

parmi lesquels un outil de feuilletage et une loupe à document (voir page 69) pour pouvoir examiner toutes les pages en un coup d'œil.

**Fig. 2.20** Un *Web Book* [Card96].

Le deuxième niveau d'organisation concerne le rangement des *Web Books*. L'utilisateur dispose pour cela de différents "meubles" pour y déposer les livres (voir la figure 2.19. Il pourra, par exemple, ranger des livres qu'il n'est pas certain de réutiliser dans un avenir proche dans les étagères alors qu'une collection de pages qu'il utilise souvent pourra rester sur le bureau. L'utilisation de telles métaphores (étagères, bureau) permet de profiter de la mémoire spatiale de l'utilisateur et de la rendre encore plus efficace en se reposant sur les habitudes de rangement que l'on peut avoir dans un bureau réel.

## Data Mountain

La *data mountain* (montagne d'informations) [Robertson98] n'est pas véritablement une visualisation du *World Wide Web*. Elle ne permet pas en effet de présenter la structure d'un site et n'a pas pour but premier la visualisation des pages HTML. Cependant, cet outil permet de ranger des pages *web* et de profiter de la mémoire spatiale pour les retrouver. C'est en fait une tentative de remplacement du système de *bookmarks*. Cette interface, illustrée dans la figure 2.21, pourrait convenir pour toutes sortes de documents mais son but initial est de gérer les pages HTML[6].

Son principe est relativement simple. L'utilisateur peut disposer les documents sur un plan incliné en utilisant une répartition convenant à sa propre mémoire spatiale. Un utilisateur préfèrera par exemple disposer les documents en fonction de leur degré d'importance du premier plan (en bas du plan incliné pour qu'ils soient toujours visibles) vers le haut du plan incliné, alors qu'un autre pourra les disposer par groupes en fonction du sujet principal qu'ils traitent. Quelle que soit la manière de les répartir, l'utilisateur profitera du fait qu'il les a placé avec sa propre logique pour les retrouver plus rapidement. Cette technique s'est avérée, selon G. Robertson *et al.*, tout à fait efficace comparée aux méthodes usuelles utilisées par les navigateurs HTML. Cependant, l'apport de la 3D n'est en rien le facteur déterminant puisqu'il a été démontré par Andy Cockburn et Bruce McKenzie [Cockburn01] qu'une telle interface en 2D était tout aussi efficace.

[6] Un système de gestion des applications quasiment identique est employé dans la *Task Gallery* [Robertson00] (section 2.4.2 page 58).

**Fig. 2.21** La *Data Mountain* ([Robertson98]).

## Autres systèmes de visualisation 3D du WWW

De nombreux projets, utilisant des techniques d'interfaces 2.5D comme celles décrites à partir de la page 67, proposent des visualisations de structures hypermédias [Andrews95, Bederson96, Gershon95b, Lagus96, Lin91, Mukherjea95, Munzner95, Munzner98]. Il serait interminable de tous les présenter ici. Une comparaison des techniques de zoom ou de vues de type *Fisheye* pour la navigation de paysages informationnels a par ailleurs été faite par Schaffer *et al.* [Schaffer96a, Schaffer96b]. En 3D aussi, il existe encore de nombreux autres logiciels de cartographie du *web* et/ou d'affichage de documents HTML [Balaguer95, Benford96, Cleary99, Cleary00, Tsou98, Fowler96, Risden00, Rubin96] ou de visualisation de trafic [Cox95] ou de gestion des signets [Rezzonico96]. Nous ne fournissons ici qu'une liste non exhaustive de quelques travaux.

Plus généralement encore, la plupart des techniques et projets présentés précédemment (qui ne sont pas faits initialement pour la présentation de cartes de sites *Web*) peuvent parfaitement être utilisés pour afficher un ensemble de liens et de pages. Il suffira pour cela de considérer que la proximité sémantique entre deux objets (deux documents hypermédias) est donnée par la longueur des liens menant de l'un à l'autre (ce qui n'est pas tout à fait exact car il n'y a pas de vérification du contenu des documents). La visualisation du contenu des pages se rapproche quant à elle de la visualisation de documents comme celle effectuée dans le Web Forager [Card96] et ne nécessite donc pas une conception sur mesure. En résumé, la visualisation d'hyperstructures convient pour l'affichage du contexte et la visualisation de documents s'applique aussi bien aux documents multimédias qu'à d'autres documents atomiques (comme une page d'un livre).

### 2.3.5 Les systèmes

La plupart des applications que nous décrivons ici peuvent être classées parmi celles capables de visualiser les hyperstructures. Cette propriété vient du fait que les composants systèmes ou logiciels ont souvent une structuration hiérarchique. Cela signifie donc qu'ils peuvent être utilisés pour la visualisation d'hypertextes, tout du moins, pour la visualisation des hyperliens à défaut de l'affichage des documents hypertextes eux-mêmes qui nécessiterait une adaptation de ces systèmes. Quoi qu'il en soit, la visualisation de composants logiciels est également un domaine à part entière qui mérite toute notre attention ; d'autant plus que ce domaine comporte de nombreux projets de visualisation 3D.

La maintenance est considérée comme étant l'activité la plus coûteuse en temps et en argent dans le cycle de vie du logiciel [Standish84]. Quelle que soit la tâche de maintenance, le programmeur doit comprendre le code initialement écrit par quelqu'un d'autre et en déduire les intentions du concepteur. Dans le meilleur des cas, il disposera d'une documentation pour apprivoiser un nouveau code source. Des outils sont donc développés pour rendre cette tâche moins laborieuse et plus efficace.

Un des domaines de recherche qui essaye de faciliter la compréhension d'un programme et, plus généralement, de rendre moins coûteuse la maintenance de logiciels, est la visualisation de logiciels. Les techniques mises en place tentent de profiter de l'habitude qu'a le cerveau humain à traiter, manœuvrer et reconnaître des images et des structures visuelles afin de réduire l'effort d'interprétation. Roman a cité dans [Roman93] un certain nombre d'arguments en faveur de la visualisation de logiciel. Ces arguments incluent notamment :
- l'importance du visuel dans la communication humaine,
- la vitesse avec laquelle les humains peuvent chercher et identifier les modèles visuels,
- le potentiel expressif d'un vocabulaire visuel qui exploite des dimensions multiples,
- et la puissance inhérente aux abstractions utilisées dans les représentations visuelles.

La crédibilité de tels arguments est souvent appuyée par le succès des représentations graphiques dans des domaines tels que les interfaces homme-machine, la visualisation de données scientifiques et la visualisation d'informations. En présentant aux utilisateurs une visualisation imagée des informations, ils peuvent se construire rapidement un modèle mental d'un programme et l'utiliser ensuite pour poursuivre plus en avant leurs investigations. La visualisation de logiciels a donc pour but de générer et de permettre la manipulation de telles structures graphiques. La lecture d'un texte (du code source par exemple) est un cas spécial de ce traitement visuel dans lequel l'interprétation se fait au niveau des caractères voire des mots. Cependant, le nombre de détails dans un texte est trop grand et le cerveau traduit naturellement chaque caractère, chaque mot ou chaque phrase en une signification ou une représentation interne. Cette abstraction est décrite dans des travaux sur la compréhension de programmes. Par exemple, les *beacons* (balises) [Brooks83] et *chunks* (morceaux) [Shneiderman80] sont des représentations internes abstraites issues des structures textuelles de bas niveau.

Mais que signifie "visualisation de logiciels" ? Cette expression n'est toujours pas très bien définie. De nombreux chercheurs ont donné leur propre définition parmi lesquelles :
- La visualisation de programmes utilise les graphiques pour illustrer certains aspects d'un programme donné sous forme textuel [Myers90].

- La visualisation de logiciels est l'utilisation des métiers de la typographie, de la conception graphique, de l'animation et de la cinématographie couplés aux technologies modernes de l'interaction homme-machine afin de faciliter la compréhension et l'utilisation efficace des logiciels [Price93].
- La visualisation de programmes est une correspondance ou une transformation d'un programme vers une représentation graphique [Roman93].
- La visualisation de logiciels schématise les processus et les données à utiliser en complément des codes sources alors que la visualisation de programmes illustre les structures de données, les états d'un programme et son code [Stasko92].

D'autres définitions encore ne s'accordent pas avec celles données ci-dessus : soit parce qu'elles se contredisent, soit parce qu'elles mettent en valeur des niveaux d'abstraction ou de détail différents. Elles mettent en évidence la variété des applications de la visualisation de logiciels. Cependant, elles semblent être en accord sur un point essentiel : la représentation visuelle qui met en évidence les liaisons entre composants logiciels (variables, fonctions, classes, méthodes). Bien que commune, cette caractérisation est toutefois très restrictive puisqu'il n'est pas déraisonnable de penser que des outils pourraient utiliser bientôt les autres sens, en particulier l'ouïe, l'odorat et le toucher.

Plusieurs chercheurs ont défini la visualisation de logiciels par rapport à d'autres expressions proches [Myers90, Price93, Stasko92]. Ces différents termes sont donnés dans la table 2.5.

La majorité des systèmes de visualisation de logiciels se sont concentrés jusqu'à récemment sur la production de représentations et d'animations bidimensionnelles de divers aspects d'un système logiciel. Les études sur l'application des graphiques 3D et des technologies de réalité virtuelle à la visualisation de logiciels sont de plus en plus nombreuses. La 3D se prête particulièrement bien au besoin toujours croissant de puissance expressive pour représenter les données, les relations multidimensionnelles qui constituent un système logiciel. Ces interfaces profitent également de l'agilité naturelle des humains pour naviguer et interagir dans les environnements tridimensionnels. La majorité des systèmes de visualisation de logiciels en 3D ne font qu'adapter des systèmes 2D [Brown93, Reiss94] en utilisant la troisième dimension pour créer des historiques ou pour afficher plus d'attributs. Ces systèmes ne profitent pas pleinement du potentiel offert par la 3D. C'est pourquoi nous ne décrivons par la suite que des systèmes qui sont véritablement 3D.

## FSN

FSN, prononcé "fusion", est un explorateur de système de fichiers développé par Silicon Graphics pour démontrer l'efficacité de leur matériel 3D. Ce projet doit sa renommée au film *Jurassic Park* dans lequel il a été utilisé. Originairement développé pour expérimenter la navigation dans des paysages informationnels, FSN se sert de la métaphore du paysage pour visualiser la structure d'un système de fichiers UNIX. Ces informations, essentiellement hiérarchiques, sont présentées comme un arbre de nœuds et de chemins plaqués sur la surface du paysage (le sol). Initialement, le point de vue de l'utilisateur est proche de la racine de l'arbre (et donc du répertoire racine). Les branches de l'arbre s'étendent en profondeur au fur et à mesure que le niveau d'imbrication des répertoires augmente.

| Expression | *Définition* |
|---|---|
| **Visualisation de calculs** (*Computation visualisation)* | L'utilisation de l'infographie pour expliquer, illustrer et montrer comment le matériel et le logiciel fonctionnent. |
| **Visualisation de logiciels** (*Software visualisation)* | L'emploi de médias visuels (et autres) pour augmenter la compréhension des systèmes logiciels. |
| **Visualisation de programmes** (*Program visualisation)* | La représentation du code source et d'autres documents pour augmenter la compréhension d'un système logiciel existant. |
| **Programmation visuelle** (*Visual programming)* | L'utilisation de représentations graphiques pour spécifier et coder les applications informatiques. |
| **Visualisation d'algorithmes** (*Algorithm visualisation)* | La représentation et l'animation des opérations abstraites des algorithmes d'un logiciel. |
| **Visualisation de codes** (*Code visualisation)* | La présentation du code réel d'un programme en améliorant son aspect textuel ou en le représentant sous une forme plus abstraite. |
| **Visualisation de données** (*Data visualisation)* | La visualisation des données définies et manipulées dans le logiciel. |

**Tab. 2.5** Les principales expressions rencontrées dans le domaine de la visualisation de logiciels.

Chaque répertoire est représenté comme un piédestal dont la hauteur est proportionnelle à la somme des tailles des fichiers qu'il contient. Les répertoires sont reliés par des chemins ou des liens qui peuvent être empruntés pour se déplacer dans la structure de fichiers. Les fichiers dans chaque répertoire sont représentés par des boîtes placées sur une grille au dessus du piédestal. Chacune de ces boîtes est habillée d'une texture plaquée sur la face supérieure qui représente le type du fichier. Comme pour les dossiers, la taille des boîtes est proportionnelle à la taille du fichier, tandis que la couleur des boîtes représente sa date de création. Grâce à l'utilisation de ce codage de l'information, couplé à la métaphore du paysage, il est facile d'identifier rapidement les grands fichiers ou répertoires. Ceux-ci peuvent alors servir de points de repère pour s'orienter dans la visualisation.

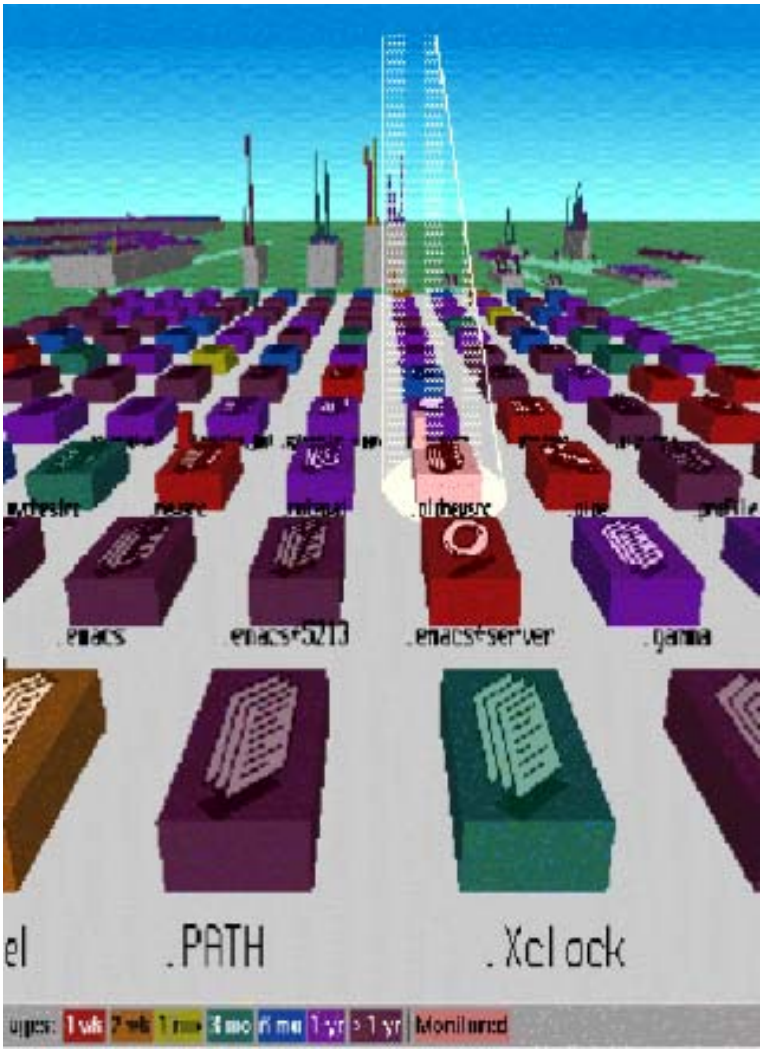


**Fig. 2.22** Visualisation d'une structure de fichiers par FSN.

Par ailleurs, FSN permet également, comme tout gestionnaire de fichiers, d'effectuer des opérations sur les fichiers et les répertoires (comme les copier ou les déplacer). De plus, certains fichiers peuvent être édités ou visualisés en exécutant des applications externes associées au type de fichier. La sélection des fichiers est effectuée grâce à un projecteur qui illumine le dossier choisi (comme on peut le voir dans la figure 2.22).

## VOGUE

VOGUE est un système, développé au laboratoire Koike, qui fournit les outils nécessaires pour créer des visualisations de logiciels en 3D. Ce système peut être utilisé dans un certain nombre de cas :
– comme une visualisation des programmes parallèles Linda (VisuaLinda - décrit ci-dessous),
– comme un outil de gestion de modules et de contrôle des versions,

– comme un explorateur de classes C++.
VOGUE est principalement employé pour créer une visualisation 3D qui intègre la gestion de module et le contrôle des versions (voir la figure 2.23 à gauche). A chaque révision, la visualisation crée un arbre bidimensionnel en fonction des dépendances entre modules. Les arbres 2D sont ensuite positionnés le long de l'axe Z, avec la version la plus récente placée au premier plan. Les rapports entre modules deviennent ainsi évidents avec cette visualisation, aussi bien pour les différentes versions d'un même module que pour les dépendances entre modules d'une même version.

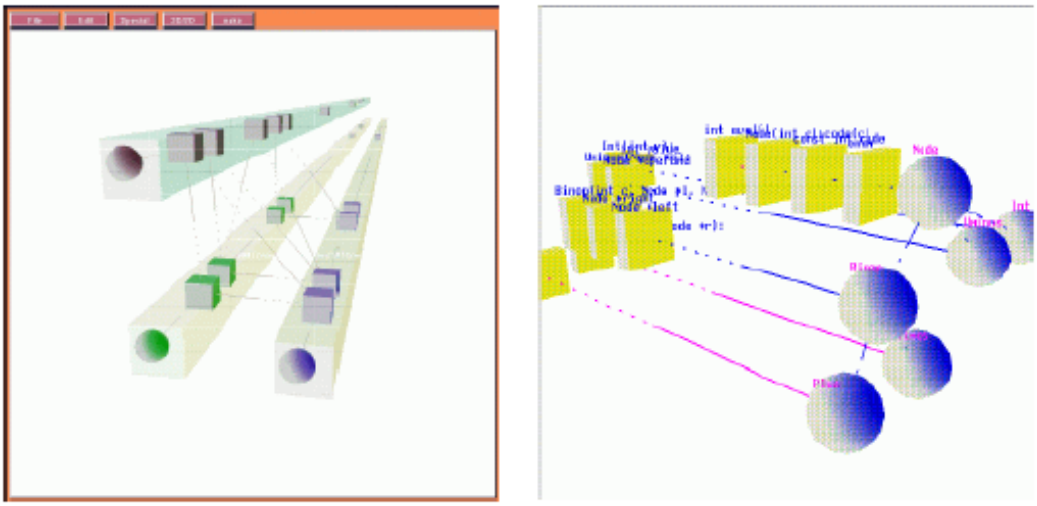

**Fig. 2.23** La gestion des versions et l'exploration de classes avec VOGUE.

L'explorateur de classes affiche deux types d'informations, tous deux dépendant de la hiérarchie d'héritage de classes (voir la figure 2.23 à droite). Le premier est la hiérarchie d'héritage proprement dite et son affichage met en valeur les classes qui dérivent d'autres classes. Le deuxième type d'informations montre les méthodes qui appartiennent à chacune des classes. Ces deux ensembles sont intégrés dans une visualisation 3D simple ayant pour but de donner immédiatement la méthode qui est exécutée réellement lorsqu'une méthode est appelée depuis n'importe quelle classe. En décalant l'orientation de son point de vue, le programmeur peut obtenir soit un affichage de l'ensemble des données, soit regarder plus particulièrement les dépendances entre les classes et les méthodes. Cette visualisation clarifie les problèmes liés à l'héritage et au polymorphisme en montrant les relations entre classes et méthodes et en n'obligeant donc pas à se construire mentalement le modèle des données.

## VisualLinda

VisuaLinda, également développé au laboratoire Koike, est un système de visualisation de programmes parallèles construit au-dessus de VOGUE. VisualLinda est un outil pour

visualiser les programmes écrits avec le langage de coordination Linda[7] [GELERNTER92] mais aussi un serveur Linda. Ces deux caractéristiques autorisent une visualisation immédiate du code sans avoir besoin d'une modification ou d'une interprétation car ceci est fait d'une manière transparente par le serveur.

La figure 2.24 illustre la visualisation obtenue lors de l'exécution de processus parallèles. Elle permet aux programmeurs de voir les messages échangés entre processus et leur évolution en fonction du temps.

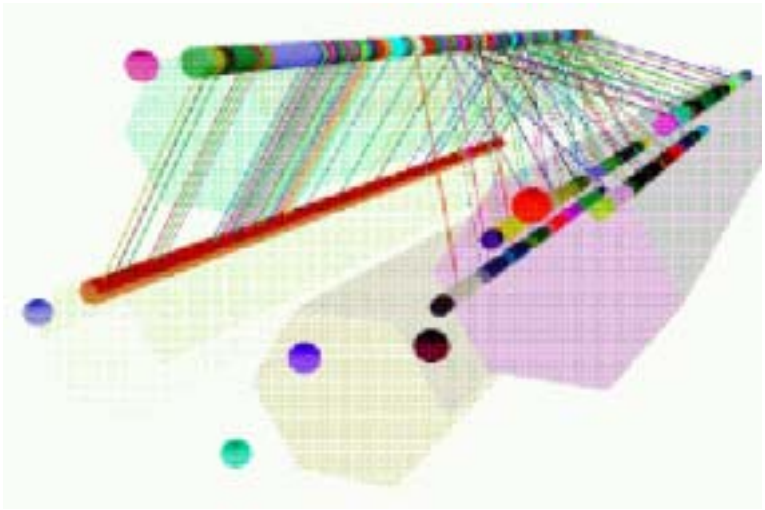

**FIG. 2.24** La visualisation de programmes parallèles avec VisualLinda.

## GRAPH VISUALIZER 3D

GraphVisualizer3D (GV3D) [WARE93] est un projet de recherche de l'université du Nouveau Brunswick dont le but est d'utiliser les technologies 3D et les environnements virtuels pour fournir une visualisation de graphes plus lisible et plus compréhensible. GV3D a la capacité de visualiser tout graphe orienté bien que son but initial ait été d'afficher les structures de logiciels. En particulier, les structures auxquelles ce projet s'attache sont les relations entre les composants dans des programmes C++ (les dépendances de modules et les relations d'héritage). Ware *et al.* décrivent les choix d'implémentation et l'architecture de GV3D et fournissent des évaluations empiriques du système et notamment une recherche sur diverses interfaces et techniques d'interaction [WARE93, WARE94]. Ces études ont évalué le nombre d'erreurs commises dans une tâche de compréhension de graphes en utilisant différentes techniques d'interaction. Le résultat le plus important dans cette étude est la diminution importante du nombre d'erreurs, et par conséquent une augmentation de la

[7] Linda permet plus particulièrement la coordination des processus par l'utilisation d'un environnement partagé.

compréhension du graphe, quand sont employés conjointement un affichage stéréoscopique et une manipulation directe du point de vue[8].

Dans GV3D, la stratégie de disposition des graphes est basée sur le regroupement sémantique plutôt que spatial. Tout comme SemNet, GV3D inclut également la notion de nœuds et d'arcs composites. Ceux-ci peuvent être ouverts ou fermés selon la volonté de l'utilisateur ou de la visionneuse (le programme en charge de la visualisation). L'ouverture d'un nœud provoque l'affichage de ses informations.

**Fig. 2.25** Plusieurs vues d'un réseau en utilisant GV3D [Ware93].

GV3D crée ses visualisations à partir d'un langage de description propriétaire. Ce langage permet de sauvegarder de manière permanente la structure du graphe, sa représentation visuelle et les informations sur la disposition des nœuds. Les données à visualiser, comme les composants d'un programme en C++, doivent donc d'abord être converties au format reconnu par GV3D. Deux méthodes principales sont possibles pour afficher la visualisation. La plus classique associe un moniteur standard et une série de *widgets* 3D pour commander la position et la vélocité des déplacements le long de chacun des axes. L'autre méthode emploie un affichage stéréoscopique dans un casque intégrant des capteurs pour les mouvements de tête. Ce dispositif fournit les indications essentielles sur la perspective qui facilitent la perception de profondeur et la reconnaissance des objets.

## Narcissus

Ce système, pensé et développé par R. Hendley [Hendley95b], produit des visualisations semblables à celles de VR-Mapper [Benford93] et de WWW3D[9] [Snowdon96]. Comme dans BEAD [Chalmers92], VR-VIBE [Benford95], Q-PIT [Benford94b] ou encore ShriMP [Storey95], des forces répulsives ou attractives agissent sur les objets. De cette manière, les objets se positionnent automatiquement dans l'espace en fonction des relations qui existent avec les autres objets. De tels systèmes, fonctionnant à base de forces, utilisent des algorithmes dits de "placement par forces dirigées" (FPS – *Force Directed Placement*). Lorsque ces algorithmes sont implémentés, les objets se déplacent dans l'espace de sorte qu'ils se rapprochent des objets avec lesquels ils sont sémantiquement liés.

[8] L'étude a également montré que l'utilisation unique de la manipulation du point de vue est plus efficace qu'un affichage stéréoscopique seul.

[9] Une des applications de Narcissus est en effet la visualisation du WWW comme nous l'avons vu lors de la présentation d'Hyperspace [Wood95] page 31.

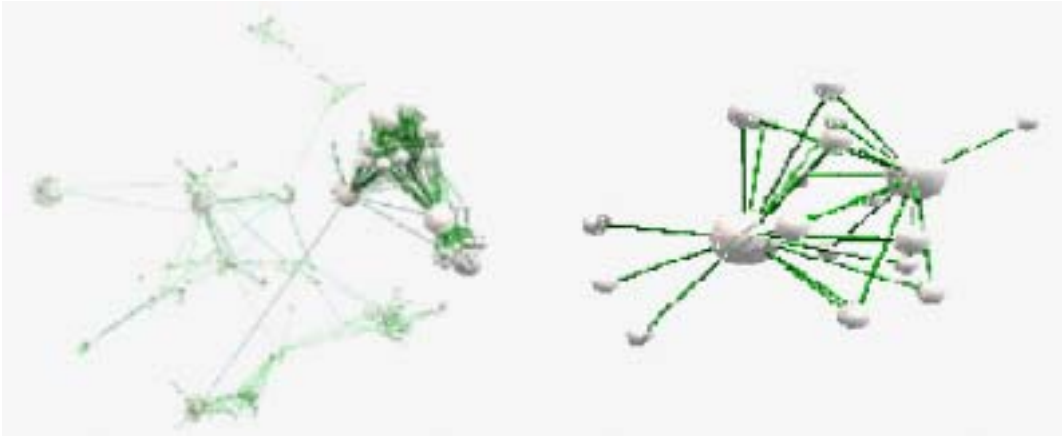

**Fig. 2.26** Représentations de structures avec Narcissus [Hendley95b].

Les structures qui émergent de ces organisations automatiques peuvent frapper l'attention et plus on travaille sur Narcissus, plus on rencontrera le même genre de structuration. Par exemple, la structure à droite dans la figure 2.26 dite du "baril" est rencontrée relativement souvent lorsque des objets (les petites sphères dans la figure) héritent tous de deux autres objets (les grandes sphères). Les objets à l'extérieur, dans cette même figure, héritent d'un seul des deux (grands) objets.

La structure à gauche dans la figure 2.26 montre un système composé de centaines de pages HTML. La partie droite dans cette figure est un ensemble des pages issues du manuel d'utilisation de Narcissus et d'exemples associés. Les grands nœuds représentent des pages HTML servant d'index vers les autres pages. La structure sous forme de boule de nœuds, au dessus, représente l'ensemble des pages du manuel depuis lesquelles il y a un certain nombre de renvois vers les exemples.

## SHriMP Views

SHriMP (*Simple Hierarchical Multi-Perspective views* – vues hiérarchiques simples utilisant la multi-perspective) [Storey95] est incorporé dans le système de *reverse ingeneering* Rigi [Storey97] pour lequel il fournit une visualisation des structures logicielles. Il est possible d'y obtenir des vues détaillées tout en maintenant une notion de positionnement ou de contexte par rapport à la structure globale du graphe. SHriMP s'appuie pour cela sur des techniques de partitionnement de graphes et sur des vues de type *fisheye* (voir dans la section 3.2.2 page 68) afin de fournir, selon les besoins, une abstraction pour le contexte et un point d'intérêt.

Le partitionnement de graphes [Harel88] est une technique employée pour soustraire des informations dans des graphes complexes afin d'essayer de clarifier leur structure. A l'inverse, ces techniques peuvent également servir à afficher les détails lorsque c'est nécessaire. De tels graphes "dépliables" sont spécifiés en considérant que les nœuds et les arcs dans le graphe peuvent être atomiques ou composés. Les nœuds composés sont visualisés comme des nœuds uniques mais contiennent un certain nombre de nœuds enfants. Pareillement,

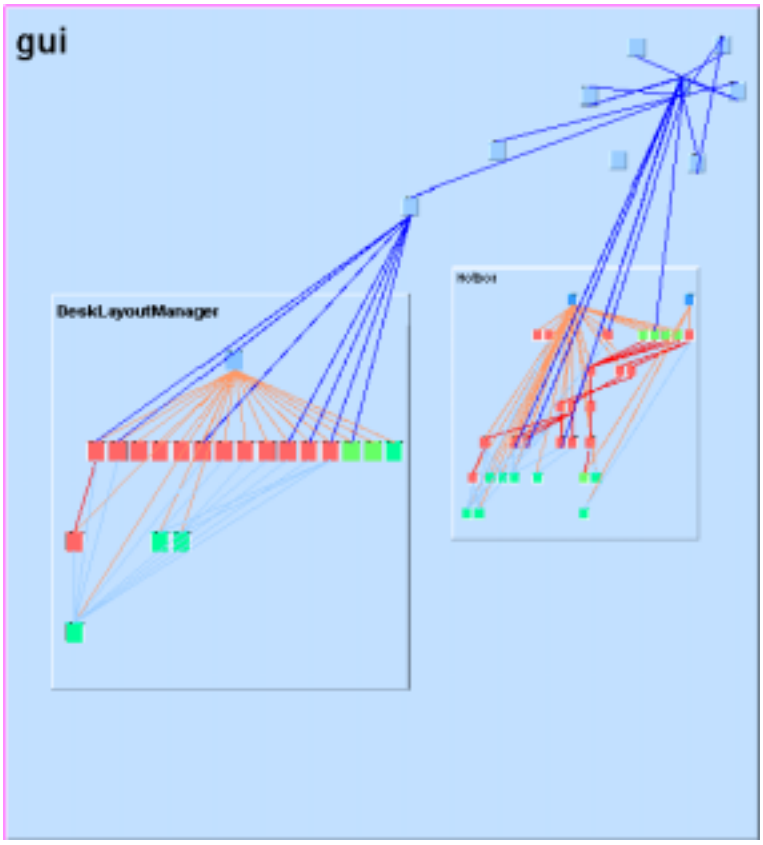


**Fig. 2.27** Une vue zoomable d'une structure logicielle avec SHriMP [Storey95].

les arcs composés peuvent représenter n'importe quel nombre d'arcs enfants. Avec de telles techniques, il est possible de simplifier visuellement des structures de graphe et l'utilisateur peut choisir librement d'afficher ou de masquer des informations.

### 2.3.6 Les ateliers de visualisation

L'atelier d'informations (ou *information workspace* ou *information visualizer*) est un concept venu du centre de recherches de Xerox à Palo Alto (PARC) et son but est de fournir un espace de travail efficace pour accéder aux informations. Ces nouveaux espaces de travail, qui profitent pleinement des graphismes 3D et de la puissance de calcul des ordinateurs récents [10] visent à remplacer à terme la métaphore du bureau et à produire une interface plus intuitive et efficace. La recherche initiale faite au PARC avait pour but d'étudier le coût de récupération d'informations dans leur espace de travail. Leur *information visualizer* [Card91] a de ce fait été créé pour favoriser l'efficacité d'organisation et la réduction au minimum des coûts de récupération des données.

[10] Voir le chapitre 3 à partir de la page 61 pour connaître les possibilités et techniques graphiques 3D des ordinateurs actuels.

L'*information visualizer* propose un certain nombre de nouvelles représentations et métaphores 3D en fonction du type de données à visualiser. Ces nouvelles visualisations incluent notamment les arbres coniques normaux et couchés [ROBERTSON91], les murs en perspective [MACKINLAY91] et les salles 3D (toutes décrites ci-après dans la section 3.2.3 à partir de la page 75). La métaphore de base de cet ensemble est celle des salles 3D, chacune des salles contenant une visualisation appropriée aux données qu'elle contient. L'*information visualizer* incorpore toutes les fonctionnalités des salles 3D décrites page 83, y compris celle de montrer une vue d'ensemble de toutes les salles en service. A partir de ce plan d'ensemble, l'utilisateur est également autorisé à manœuvrer les informations des salles (avec considérablement moins de détail).

La navigation dans les salles est faite à travers un *avatar* humain que l'on peut orienter indépendamment des mouvements effectués de manière à reproduire un déplacement réaliste d'un humain (qui se déplace dans une direction tout en ayant la possibilité de regarder ailleurs). Les interactions se composent de fonctions classiques comme examiner les objets mais aussi par des fonctions plus avancées comme indiquer des opérations par des mouvements de souris. Les objets ou le point de vue peuvent être déplacés librement (marcher vers les objets ou amener les objets devant soi). Lors d'un déplacement, la vitesse est logarithmique assurant un contrôle maximal du positionnement. Beaucoup d'éléments dans l'*information visualizer* répondent aux gestes. Ces gestes sont interprétés à travers les mouvements de la souris (par exemple, déplacer la souris dans une direction ou tracer un motif de croix). Les gestes sont une méthode simple pour exécuter des commandes sans libérer la souris ou exiger la présence de nombreuses icônes de fonction. Ce principe a d'ailleurs été repris par R. Zeleznik [ZELEZNIK99] pour contrôler les mouvements du point de vue en utilisant uniquement une souris 2D.

## 2.4 Autour de nos préoccupations

Outre les domaines vus précédemment, la visualisation d'informations et de documents s'applique à deux familles d'applications de manière particulièrement naturelle. Pour chacune d'elles, les bibliothèques numériques et les interfaces graphiques associées aux systèmes d'exploitation, nous pensons que l'utilisation d'une interface 3D est une alternative intéressante face aux interfaces couramment utilisées pour y accéder. Ces deux domaines ayant été le champ de nos expérimentations, nous y consacrons deux sections distinctes. Nous présentons donc ici, pour chacun d'eux, les principaux travaux, projets et études réalisés.

### 2.4.1 Les bibliothèques numériques

Les bibliothèques numériques, également appelées bibliothèques électroniques, ont attiré l'attention des chercheurs en informatique et des industriels depuis la fin des années 80. Les avancées technologiques des systèmes informatiques et des télécommunications rendent aujourd'hui leur développement possible. Les bibliothèques numériques sont, en effet, très liées aux réseaux, au multimédia et aux technologies hypertextes et hypermédias. Cet

aspect pluridisciplinaire en fait un domaine dont les frontières sont mal définies. Cependant, connaissant les bibliothèques traditionnelles, on peut donner la définition suivante faite par Fox [Fox94] :

> *A Digital Library is a machine readable representation of materials which might be found in a university library together with organizing information intended to help users find specific information. A Digital Library service is an assemblage of digital computing, storage, and communications machinery together with the software needed to reproduce, emulate, and extend the services provided by conventional libraries based on paper and other material means of collecting, storing, cataloguing, finding, and disseminating information. A full service digital library must accomplish all essential services of traditional libraries and also exploit digital storage, searching, and communication.* [11]

Les bibliothèques numériques ne sont pas exclusivement des programmes informatiques qui permettent d'accéder à des documents en ligne. Plus généralement, on peut les définir comme étant des organisations qui sélectionnent, organisent et garantissent l'accès à des documents. Ces tâches nécessitent les mêmes métiers que ceux des bibliothèques traditionnelles à l'exception de la mise à disposition des ouvrages qui se fait par un personnel technique spécialisé.

### Activités et caractéristiques

Une bibliothèque traditionnelle peut être décrite comme un ensemble de quatre activités différentes associées à un ensemble de tâches particulières. Ces activités sont données dans le tableau 2.6. Dans les bibliothèques numériques, les mêmes tâches sont présentes mais sont, dans la mesure du possible, automatisées.

Il est bien évident que l'essor d'Internet a fait le succès des bibliothèques numériques puisqu'on y accède à distance. Les documents récupérables à travers le réseau forment une bibliothèque universelle, accessible de n'importe où et par n'importe qui. D'autre part, les documents peuvent être distribués physiquement sur plusieurs machines plutôt que centralisés au même endroit. Ainsi, on peut considérer les bibliothèques numériques comme un environnement réparti qui facilite énormément le partage, la réutilisation, le stockage et l'intégration. Les caractéristiques plus générales d'une bibliothèque numérique sont issues de la liste (non exhaustive) des domaines suivants :
- les bases de données multimédias,
- la recherche d'informations,
- la visualisation d'informations,

[11] Une bibliothèque numérique est une représentation interprétable par une machine de matériels que l'on peut trouver dans une bibliothèque universitaire avec des informations d'organisation dont le rôle est d'aider les utilisateurs à trouver une information spécifique. Une bibliothèque numérique est un mélange d'opérations numériques tels que les calculs, le stockage et les communications avec le logiciel requis pour reproduire, émuler, et étendre les services offerts par les bibliothèques traditionnelles basées sur le papier et d'autres matériaux ; services qui sont le regroupement, le stockage, le catalogage, l'interrogation et le partage de l'information. Une bibliothèque numérique doit accomplir tous les services qu'accomplissent les bibliothèques classiques mais aussi exploiter le stockage, la recherche et la communication.

| **Activité** | *Description* |
|---|---|
| **Collection** | Ensemble de techniques pour déterminer quelles ressources sont accessibles. |
| **Organisation et représentation** | Classification et indexation des ressources d'une manière à les retrouver plus facilement. |
| **Accès et récupération** | Organisation de l'espace physique et répartition des ressources dans cet espace pour répondre efficacement aux attentes des utilisateurs. |
| **Analyse et synthèse** | Ces activités servent à produire des documents de référence pour répondre à des questions précises. |

**Tab. 2.6** Les quatre activités d'une bibliothèque.

- les interfaces homme-machine,
- la digitalisation de documents,
- la retouche d'images,
- la reconnaissance de caractères,
- la création de sites web,
- le commerce électronique.

## Les précurseurs

Les premières bases de données de documents numériques n'avaient pas pour but de faciliter l'accès à des ouvrages, mais de réduire les coûts de production dans l'imprimerie. On trouvait en effet des documents digitaux, dès la fin des années soixante, chez les éditeurs qui souhaitaient rééditer rapidement certains de leurs ouvrages.

La première machine électronique pour restituer des informations a été pensée en 1945 par Vannevar Bush du MIT. Cette machine, du nom de MEMEX (*MEMory EXtension*), devait suppléer la mémoire humaine par des moyens électroniques afin de rendre facilement accessibles toutes les connaissances humaines et pallier l'inefficacité des outils traditionnels. Le MEMEX que l'on peut voir dans la figure 2.28 aurait, en théorie, pu stocker des milliards d'informations et les restituer à l'utilisateur sur simple demande. Ce dernier aurait eu la possibilité d'opérer à distance à l'aide d'un clavier et aurait fait apparaître les informations sur un écran grâce à des liens[12]. Bien que cette machine n'ait jamais été réalisée, elle a inspiré de nombreuses études sur les bibliothèques numériques et les hypertextes. En découlèrent, notamment, les travaux de W. Weaver sur les traducteurs automatiques pour rendre accessibles à tous des ressources écrites dans une langue étrangère, l'invention de l'hypertexte par Ted Nelson en 1960 ou encore l'apparition du *World Wide Web* grâce aux travaux de Tim Berners-Lee.

[12] On notera que le MEMEX fut aussi un précurseur dans le domaine des documents hypertextes.

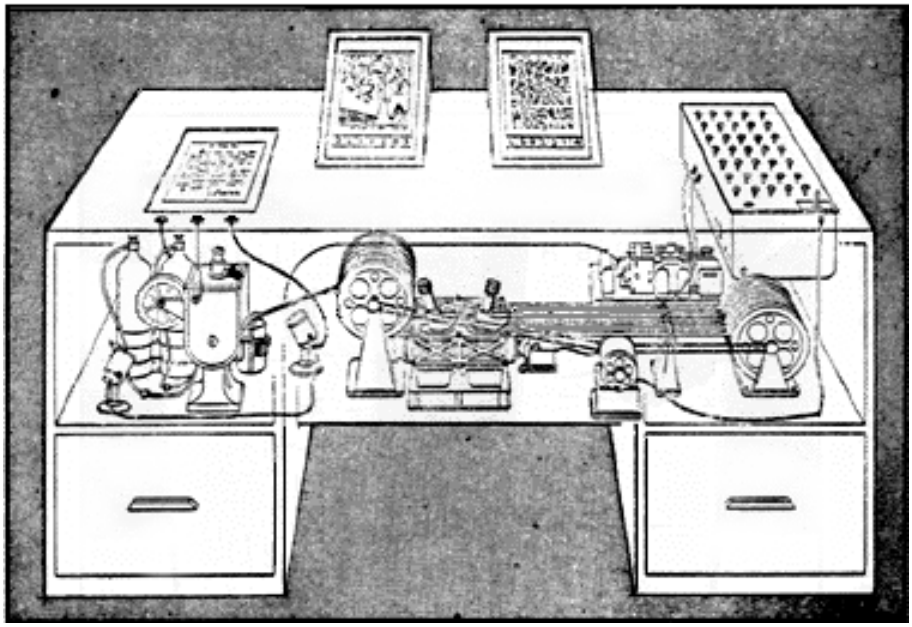

**Fig. 2.28** Le MEMEX de Vannevar Bush.

Outre le MEMEX, Vannevar Bush imagina également de nouveaux systèmes pour faciliter le travail sur les documents. La figure 2.29 illustre deux dispositifs d'entrée. A gauche, il s'agit d'une machine avec une interface vocale pour interroger une base documentaire. A droite, on voit un homme portant une mini-caméra pour enregistrer ces expérimentations afin de les conserver et d'y accéder par la suite.

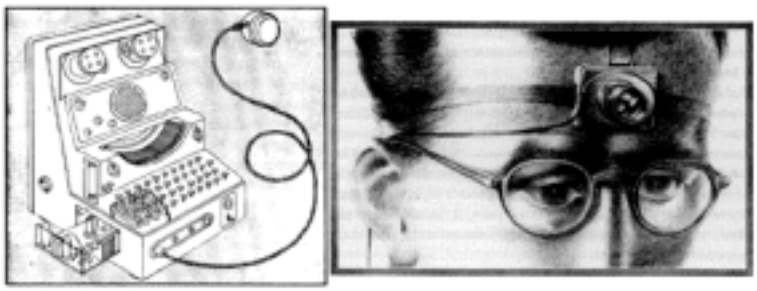

**Fig. 2.29** Les dispositifs d'entrée imaginés par V. Bush.

Chacun des travaux précédents se rejoignent sur le principe de rendre les documents accessibles au plus grand nombre avec un moindre effort. Bien que l'idée du MEMEX soit merveilleuse, plusieurs problèmes se posaient quant à sa réalisation. En effet, concevoir ce qui devait plus tard devenir une bibliothèque numérique n'est pas uniquement la transformation et la compilation sur des supports magnétiques de milliers d'ouvrages, de journaux, de notes et d'images. C'est avant tout repenser les liens qui doivent s'opérer entre l'utilisateur et la machine, imaginer des outils de recherche efficaces et intuitifs et entreprendre un changement radical dans le rapport avec l'objet physique qu'est le livre. Pendant long-

temps, la puissance des ordinateurs et l'absence d'interface graphique ne permirent pas d'envisager un tel système.

Dans ce contexte, les premiers travaux se sont naturellement consacrés à l'amélioration de l'indexation, de la recherche et de mise en rapport (par hyperliens) des documents plutôt qu'à la création d'un poste de lecture. Ces recherches dont le but était de permettre aux utilisateurs de trouver rapidement un document ont abouti à la création de plusieurs normes telles que MARC (*MAchine Readable Cataloging*), SGML (*Standard Generalized Markup Language*), CALS (*Continuous Acquisition and Life-Cycle Support*), TEI (*Text Encoding Initiative*) ou encore HTML (*HyperText Markup Language*). Les premiers projets de bibliothèques numériques en mode texte voient cependant le jour dans les années soixante-dix, parmi lesquels celui de Michael Hart, Guttenberg. Il visait à mettre à disposition du public des ouvrages saisis en ASCII par des volontaires passionnés. Cette idée de volontariat a également été appliquée à l'une des premières bibliothèques numériques françaises, ABU (L'Association des Bibliophiles Universels).

L'apparition de l'interface graphique de Xerox, popularisée par Apple et son Macintosh, a permis d'envisager de nouvelles manières pour présenter des documents, autres que l'utilisation exclusive de caractères ASCII. Ces interfaces ont permis de présenter non plus des documents au format texte mais des fac-similés de documents dans lesquels sont visibles aussi bien les textes que les illustrations. Or, ce mélange de textes et d'images pose de nouveaux problèmes que J. André, J.-D. Fekete et H. Richy ont soulevés [André95]. Par exemple, ce format graphique nettement plus riche visuellement ne permet pas de faire des recherches d'occurrences sans adjonction de systèmes de reconnaissance de caractères. De plus, la recherche d'illustrations et de lettrines, particulièrement importantes pour bon nombre de lecteurs passionnés, n'est pas triviale. Sur ces points, on peut noter le travail de la société iArchives (`www.iarchives.com`) qui propose un format de fichier associant l'image du document avec des informations sur sa structure. Il permet de retrouver certains détails tels que les illustrations, les signatures ou encore les typographies tout en offrant quelques fonctionnalités de manipulation des documents digitalisés.

## Les recherches en cours

Alors que les problèmes de digitalisation, de capacités de stockage, de coût d'utilisation et de puissance disponible sont en partie résolus, d'autres aspects des bibliothèques numériques monopolisent aujourd'hui l'attention des chercheurs. Nous les présentons ici mais nous ne nous attarderons que sur les problèmes liés à la présentation des collections et des documents dans les sections suivantes.

### *Un autre Web*

Les systèmes interactifs, en particulier ceux fédérant une base de données d'informations multimédias, sont au cœur des bibliothèques numériques émergentes. Dans ces bibliothèques récentes, les problématiques principales sont la recherche de documents et la liaison entre les différentes informations. Et pour celles-ci, il est fort naturellement préconisé de les rendre aussi transparentes et efficaces que possible.

E. Fox *et al.* [Fox93] conviennent que les informations multimédias peuvent être présentées sous forme digitalisée sans casser la chaîne numérique pour une étape qui requerrait une conversion analogique. Depuis les documents eux-mêmes, en passant par la description de leur contenu en respectant un format standard international, jusqu'à leur organisation utilisant les capacités des réseaux hypertextes et hypermédias, toutes les informations d'une bibliothèque peuvent circuler sous forme numérique. Cependant, pour atteindre ce but, il est nécessaire de mettre au point une architecture unifiée et fédératrice capable de regrouper les informations préservées sur des sites distants et hétérogènes. Fox *et al.* se penchent sur ce problème particulier nécessitant d'étudier non seulement la gestion distribuée des connaissances mais également, des protocoles réseaux et normes hypertextes mieux adaptés à ces fins.

Il est évident que les technologies, les infrastructures et les protocoles réseaux du *World Wide Web* ont permis la multiplication des ressources accessibles. Pour exemple, les entreprises éditent leur catalogue en ligne, les organisations gouvernementales proposent un certain nombre d'informations, une multitude de groupements d'intérêt mettent à disposition de nombreuses informations, des journaux, des magazines, des livres et des rapports sont également publiés sur l'Internet. Le rythme auquel le nombre d'informations évolue n'a, pour l'instant, aucune raison de baisser.

L'utilisation d'un langage commun pour accéder aux données a grandement facilité cette multiplication de ressources. TCP/IP est la couche basse commune de tout échange sur Internet. Le protocole HTTP, véhiculant les requêtes et les réponses entre clients et serveurs *web*, est également un dénominateur commun. Enfin, ces sites sont construits en utilisant un nombre réduit de langages (HTML, Javascript, Java, Flash, VRML, ...) et testés sur les clients les plus utilisés (Internet Explorer, Netscape Navigator, ...) pour conserver un auditoire le plus large possible. Cependant, pour gérer, préserver, regrouper, rechercher et accéder aux données réparties, aucune méthode standardisée n'est offerte. Or ces tâches sont particulièrement importantes dans les bibliothèques numériques. C'est pourquoi de nombreux efforts visent à définir une technologie commune pour permettre une interopérabilité entre les nombreuses bibliothèques numériques.

### *Les informations hétéroclites*

Un autre point important dans les bibliothèques numériques est la gestion des documents issus de sources différentes. De nombreuses questions se posent quant à la coexistence et le rapprochement de deux collections différentes. L'intégration, par exemple, des documents papiers et électroniques dans une même bibliothèque pose encore un problème. Pour les documents papiers, dans le meilleur des cas, il existe des fiches qui peuvent être intégrées à la bibliothèque électronique. Mais comment intégrer les documents papiers pour les mettre en valeur au côté des documents numériques ? Quelle politique éditoriale suivre pour conserver un intérêt pour les documents papiers face aux collections numériques grandissantes ? En attendant qu'ils soient digitalisés, comment les rendre accessibles à des utilisateurs distants de la même manière qu'ils peuvent accéder aux documents numériques ?

Les recherches sur les bibliothèques numériques se concentrent jusqu'à maintenant sur les documents institutionnels. Une autre source d'inquiétude concernant les ressources hété-

rogènes vient des productions individuelles ou de groupes non gouvernementaux. Outre le problème très général d'archivage posé par la masse de documents que nous produisons tous et qui risquent de se perdre à cause des changements de format de fichier ou de support, il se pose un autre problème : comment intégrer les informations dignes de l'être dans une bibliothèque numérique ? Ces producteurs de documents d'analyse ou de synthèse, d'index, de taxonomies, de réponses à des questions souvent posées utilisent parfois eux-mêmes des bibliothèques numériques. Il serait donc intéressant d'intégrer ces productions pour accroître la qualité des informations accessibles. Plus globalement, cela pose aussi le problème du partage d'annotations de documents.

## La visualisation

Pour les bibliothèques numériques, une interface est nécessaire pour avoir accès à plusieurs aspects : la présentation de la collection, la sélection et la lecture des ouvrages et leur annotation. La plupart des projets de visualisation, quel que soit l'aspect auquel ils s'attachent, ont été initiés pour faciliter l'accès aux livres. Certains s'appuient sur des métaphores de navigation et de représentation réelles pour profiter de l'expérience des utilisateurs alors que d'autres proposent de nouvelles métaphores capables d'offrir de nouveaux services.

Pour les postes de lecture, tous les projets buttent sur un problème de taille qu'est la lecture sur écran. En effet, la lecture sur un moniteur n'offre pas d'avantages, bien au contraire, par rapport à la lecture papier. La majorité des lecteurs sont gênés par les contrastes des supports informatiques et par la nécessité de devoir faire défiler verticalement le document. Gould *et al.* soulignent bien ce rapport entre la représentation et la vitesse d'acquisition des informations [Gould84]. Pour les documents numériques, ils constatent que la résolution du dispositif de sortie est un élément déterminant dans la vitesse de lecture [Gould87]. De leur côté, Jones et Dumais jugent que la plupart des interfaces documentaires jusqu'alors proposées offrent de nombreux avantages mais ne permettent pas à l'utilisateur de se souvenir de l'endroit où un ouvrage a été trouvé [Jones86].

Dans son ouvrage sur les bibliothèques numériques, Mickael Lesk [Lesk97] précise que la majorité des utilisateurs préfèrent "fureter" dans un fonds d'ouvrages plutôt que d'y effectuer une recherche directe. Dans la plupart des cas, les utilisateurs n'ont pas systématiquement une idée précise des ouvrages qu'ils cherchent mais plutôt une idée du sujet qui les intéresse. Dans ce cas, ils parcourent un ensemble d'ouvrages visuellement puis, par éliminations successives, sélectionnent quelques ouvrages répondant à leurs critères de choix initiaux.

Dans ce contexte, le choix d'une interface graphique 3D peut être judicieux car il est certainement plus facile d'y intégrer ces heuristiques visuelles. Cependant, l'inconvénient majeur dans la représentation en 3D d'un important volume d'informations est le phénomène de désorientation de l'utilisateur. Celui-ci peut rapidement se trouver dans une situation où il lui est difficile de se localiser. Ces problèmes sont très souvent rencontrés lorsque l'utilisateur a la liberté de naviguer à travers une multitude de documents sans avoir une vision globale du contexte. Certaines études ont tenté de proposer des solutions à ce problème de spatialisation et de désorientation. L. Robert et E. Lecolinet [Robert98] proposent un système de représentation hiérarchique multi-fenêtres de documents HTML parcourus lors

d'une navigation sur le Web. En complément, S. Pook *et al.* [Pook00] proposent des outils d'aide à la navigation permettant à l'utilisateur de connaître à tout instant sa position par rapport au contexte global.

Une initiative intéressante a été réalisée avec LibViewer [Rauber99a] qui offre plusieurs heuristiques visuelles 2D pour rendre plus intuitives les recherches et la représentation des ouvrages dans un important fonds documentaire (voir la figure 2.30). Ce développement repose également sur un autre outil intéressant, SOMLib, qui permet une organisation automatique des catalogues [Rauber99b]. Dans ce système, les auteurs ont avant tout souhaité donner à chaque élément des caractéristiques particulières afin d'offrir aux utilisateurs des repères visuels directs. Chaque type d'objet (journal, magazine, manuscrit, enregistrement audio, etc.) est représenté différemment. Les couleurs sont utilisées pour informer l'utilisateur de certaines spécificités de l'objet (langage, genre etc.).

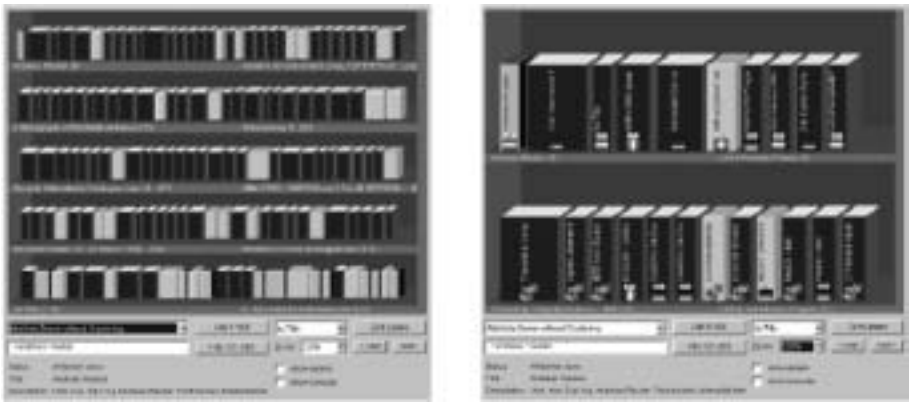

**Fig. 2.30** L'interface de libViewer [Rauber99a].

Les interfaces 3D pour accéder aux bibliothèques numériques sont apparues relativement tard. Le manque d'un matériel peu coûteux mais performant en est la cause. S'agissant d'un domaine d'application relativement récent de la 3D, les études ne sont pas nombreuses. La plupart proposent des représentations virtuelles de la véritable bibliothèque [Fox97, Plénacoste01]. Les projets s'attachant à une représentation métaphorique des bibliothèques sont quasi inexistants. On peut toutefois citer l'interface du WebForager [Card96] (voir page 36), qui bien que réalisée pour la collecte d'informations sur Internet, révèle de nombreuses possibilités pour les bibliothèques numériques. Et en étudiant de telles interfaces, on comprend tout à fait que les heuristiques visuelles ont le pouvoir de simplifier des tâches cognitives complexes par l'utilisation du système perceptif [Hearst96]. Dans cet article, les auteurs ont utilisé les visualisations de type *Cone Trees* (voir page 81) et *Perspective Wall* (voir page 80) pour créer un explorateur de citations.

### 2.4.2 Les bureaux 3D

Un autre domaine notable dans la visualisation de documents, auquel nous nous sommes particulièrement intéressé (voir la section 4.5 page 174), est la représentation en 3D d'ap-

plications. Nous parlons ici plus précisément des projets d'interfaces 3D visant à remplacer la métaphore du bureau traditionnel. Leur but n'est pas de remplacer les *widgets* 2D avec lesquels sont construites les interfaces des applications, mais plutôt d'intégrer ces interfaces 2D dans un environnement 3D.

Devant l'augmentation constante des logiciels exploitant une interface 3D (comme les modeleurs, les navigateurs, les jeux), nous ne cessons de passer d'un contexte 2D à un contexte 3D. Ce basculement d'un environnement à l'autre change en premier lieu la vision que l'on a des objets : en 2D ils apparaissent plats alors qu'en 3D ils ont un volume. Mais avec ce changement d'interfaces, s'opère également une modification des méthodes d'interactions. Plutôt que de présenter les applications 3D dans des fenêtres 2D, l'idée est donc d'inclure les fenêtres des applications 2D dans un environnement 3D. De cette manière, les logiciels pourront coexister sans devoir changer de méthode d'interaction. On manipulera les objets 3D et les fenêtres 2D de la même manière puisque ces dernières bénéficieront des mêmes libertés spatiales que tous les autres objets 3D.

En second lieu, les bureaux 3D peuvent apporter une solution au problème des fenêtres recouvrantes : la multiplication des applications ouvertes obscurcit l'espace de travail. Nous présentons ici les quelques projets de bureaux 3D qui ont retenu notre attention. Tous ne permettent pas de représenter à la fois des objets 3D et des fenêtres 2D. Cependant, ils contribuent à créer l'émulation autour de ce sujet.

## MaW3D

La métaphore employée dans MaW3D [Leach97] pour organiser les fenêtres des applications 2D est celle du tunnel, comme on peut le voir sur les copies d'écran de la figure 2.31. L'utilisateur est placé à l'entrée du tunnel de section rectangulaire, en son centre, et regarde vers l'autre extrémité. Le tunnel et les fenêtres qui y sont accrochées sont affichés par une projection en perspective. Les fenêtres sont essentiellement des éléments 2D du fait que leur zone de travail est plane. Elles disposent cependant de décorations en 3D.

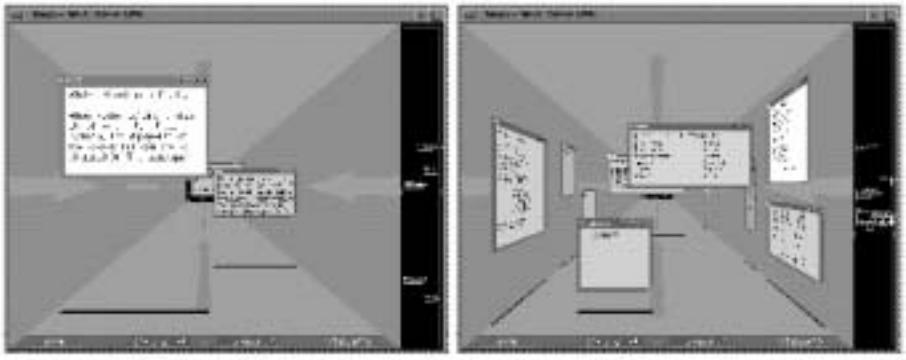

**Fig. 2.31** La métaphore du tunnel de MaW3D [Leach97].

Les fenêtres peuvent être placées à une profondeur arbitraire dans le tunnel mais elles sont contraintes au niveau de leur orientation. Elles sont obligatoirement orthogonales à l'axe du tunnel. Lorsque l'utilisateur pousse une fenêtre dans le tunnel, elle reste perpendiculaire aux faces du tunnel et sa taille est automatiquement réduite. Elle est calculée de manière à être inversement proportionnelle à la distance séparant la fenêtre de l'utilisateur. Les fenêtres sont également contraintes de sorte qu'au moins une partie demeure toujours à l'intérieur du tunnel. Une fenêtre pourra ainsi être descendue suffisamment afin que sa barre d'état soit visible, comme dans la copie d'écran à gauche dans la figure 2.31. Cette propriété s'avère pratique à la fois pour conserver un indice sur la fenêtre cachée mais aussi pour désencombrer le tunnel. Il est également possible d'accrocher les fenêtres sur les murs gauches et droits du tunnel, comme cela est représenté dans la copie d'écran à droite dans la figure 2.31. Les fenêtres peuvent soit être accrochées individuellement soit toutes de manière immédiate. Cette dernière fonctionnalité permet à l'utilisateur de déterminer rapidement où se trouvent les fenêtres dans le tunnel.

Il existe enfin deux dernières fonctionnalités majeures dans cette interface. La première est un plan du tunnel vu du dessus, qui se trouve sur la droite de l'écran. Les fenêtres peuvent être sélectionnées et manipulées directement à partir de ce plan. La deuxième fonctionnalité est une console qui se trouve en bas de l'interface. C'est dans cette console que l'on accède aux outils de paramétrages et aux différentes fonctionnalités.

## 3DWM

3DWM (*3D Workspace Manager*) est un projet de Chalmers Medialab, disponible gratuitement[13], visant à produire un environnement 3D. Ce n'est pas un système de fenêtrage X11 mais une interface entièrement 3D visant à le remplacer. Ce projet est le premier à permettre la coexistence des applications 2D et 3D. Pour cela, les fenêtres des applications 2D sont utilisées comme textures de faces 3D. Plus exactement, le *frame buffer* (le tampon conservant l'image de l'interface) est utilisé comme texture. Cela implique donc que la totalité de l'ancien bureau 2D soit plaqué sur une seule face, comme on peut le voir sur la figure 2.32.

3DWM, plus qu'une simple interface 3D, est une architecture client/serveur s'appuyant sur CORBA. Grâce à CORBA, les clients peuvent être écrits avec différents langages et exécutés sur des machines diverses. Le module Garbo s'occupe spécifiquement de la gestion des applications 2D. Il a été implémenté avec le système d'affichage distant VNC (*Virtual Network Computing*) développé au laboratoire AT&T de Cambridge [Richardson98]. L'utilisation de VNC apporte deux limitations. La première, nous l'avons cité plus haut, est la récupération de tout le *frame buffer*. La deuxième est de devoir toujours exécuter son gestionnaire de fenêtre 2D en dessous de l'environnement 3D. N. Roussel a également utilisé VNC dans son projet d'environnement 3D [Roussel01] mais a fait en sorte que les fenêtres ne se chevauchent pas pour pouvoir extraire leur image individuellement.

[13] 3DWM, ses documentations et son tutorial peuvent être téléchargés sur http ://www.3dwm.org.

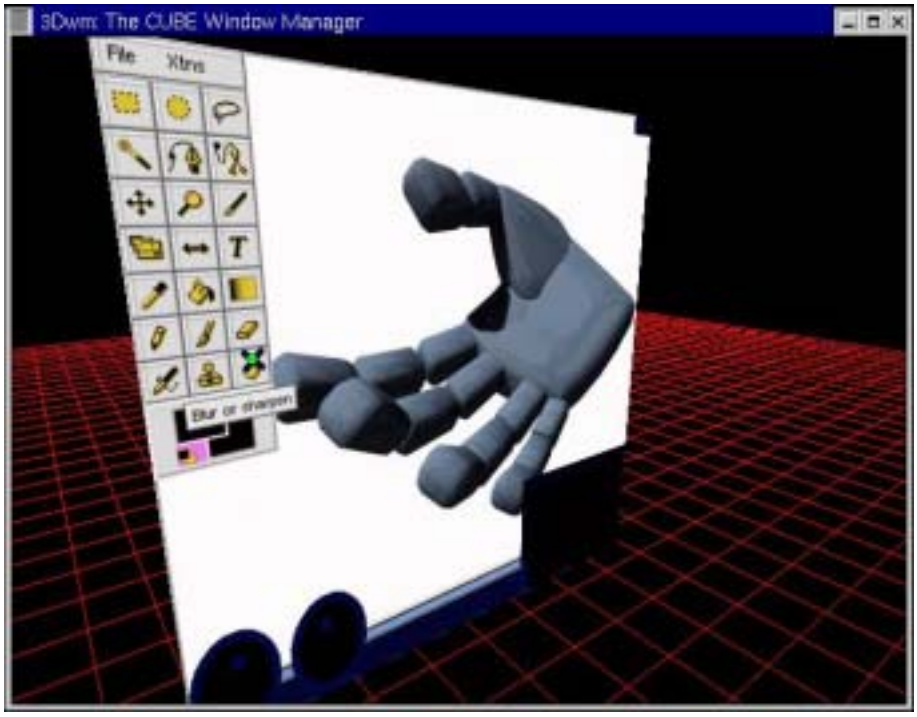


**Fig. 2.32** Le gestionnaire de fenêtres 3DWM.

## Task Gallery

La TaskGallery [Robertson00] (littéralement "la galerie de tâches") est un projet d'interface utilisateur en 3D pour le système d'exploitation Windows de Microsoft. Disposant de la puissance de développement de la société Microsoft, ce projet est de loin le plus avancé. Son but est d'augmenter le bureau traditionnel de Windows en un véritable environnement 3D dans lequel peuvent être gérés une multitude de bureaux. La métaphore principale utilisée dans la TaskGallery, comme son nom l'indique, est celle de la galerie d'art. Cette métaphore est très proche de celle du tunnel de MaW3D comme on peut le voir dans la figure 2.33. L'utilisateur se déplace à l'intérieur d'un couloir sur les murs duquel sont posées des "peintures" représentant les différents bureaux contenant les applications. L'utilisateur peut se déplacer rapidement d'un tableau à l'autre par des mouvements simples initiés à la souris et au clavier. L'utilisation d'un environnement aussi contraint, puisque l'utilisateur ne peut pas évoluer hors de ce couloir, réduit le nombre de degrés de liberté nécessaire et permet donc d'utiliser les dispositifs d'entrée usuels. De plus, conserver les dispositifs 2D classiques permet d'envisager l'installation de cet environnement par la majorité des utilisateurs.

Les outils mis à disposition pour se déplacer et interagir avec les tableaux sont d'une grande simplicité afin qu'ils conviennent à tous les utilisateurs et non pas uniquement aux *hardcore gamers* (aux joueurs invétérés). A tout moment, se présente dans la coin inférieur gauche de l'écran les options de navigation (voir la copie d'écran à gauche dans la figure 2.34).

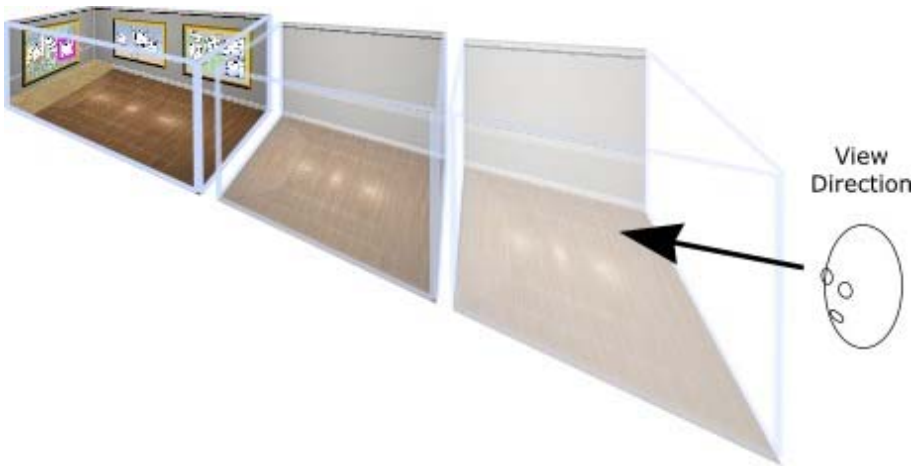


**Fig. 2.33** La métaphore du tunnel de la TaskGallery [Robertson00].

Il suffit à l'utilisateur de cliquer sur l'un des *widgets* illustrés à droite dans la figure pour effectuer l'opération.

**Fig. 2.34** Les options de navigation dans la TaskGallery [Robertson00].

Au niveau interaction, un certain nombre de fonctionnalités sont offertes. Il est possible de regarder, en gros plan, un tableau particulier. C'est le cas dans la figure 2.34. L'interface ressemble et s'utilise pratiquement comme une interface de bureau classique à l'exception près, comme on peut le constater, que les fenêtres peuvent être rangées en utilisant des

politiques de placement 3D. Un outil est également offert pour conserver toujours à portée de main des applications particulières, celles que l'utilisateur utilise le plus souvent par exemple. Ainsi, il peut les rappeler rapidement au lieu de se souvenir des tableaux dans lesquels elles se trouvent. L'expression "à portée de main" est à prendre au sens propre puisque ces applications sont accessibles par une palette (figure 2.35).

**Fig. 2.35** La palette d'applications de la TaskGallery [Robertson00].

Le dernier point important est le positionnement des tableaux dans la galerie. L'utilisateur a la liberté de les accrocher sur n'importe quel mur de n'importe quelle section de la galerie. Comme on peut le voir dans la figure 2.36, il est également possible d'utiliser le sol. Le déplacement des tableaux se fait par simple glisser-déposer et le programme gérant l'interface exécute les calculs nécessaires pour que les tableaux se positionnent correctement.

**Fig. 2.36** Le positionnement des bureaux dans la TaskGallery [Robertson00].

# Chapitre 3

# *Techniques et technologies*

**Contenu du chapitre**

*Ce chapitre présente les techniques et technologies utiles pour la visualisation d'informations et de documents. Ces techniques, mises en œuvre dans les applications décrites dans le chapitre 2, sont ici classées en fonction du nombre de dimensions utilisées. Sont présentées, en particulier, les techniques 3D employées pour afficher de nombreuses données structurées. Pour celles-ci, nous détaillons les technologies et les outils 3D les plus couramment utilisés.*

*Ce chapitre est organisé de la manière suivante :*


- Section 3.1 – Introduction du chapitre – p. 62 : *Constat que nous pouvons faire de l'utilisation des interfaces WIMP.*
- Section 3.2 – Interfaces et dimensions – p. 63 : *Classification des "nouvelles" interfaces en fonction du nombre de dimensions utilisées.*
- Section 3.3 – Avancées technologiques – p. 86 : *Présentation des technologies 3D, depuis la couche matérielle jusqu'à la couche application.*
- Section 3.4 – Evolution du matériel – p. 119 : *Présentation des évolutions des domaines liés à la 3D. Nous étudions notamment les dispositifs de pointage et le son spatialisé.*

## 3.1 Introduction du chapitre

Durant leur trente premières années, les interfaces utilisateur ont connu au moins trois changements principaux en passant par les *batchs*, les lignes de commande et les interfaces WIMP. Depuis lors, nous utilisons toujours la GUI (*Graphical User Interface*) issue du paradigme WIMP. Alors que ces interfaces ont marqué un énorme pas en avant par rapport aux lignes de commande, elles semblent, aujourd'hui, ne pas être en adéquation avec la puissance offerte par nos ordinateurs et le nombre de données à manipuler. Ceci ne doit pas être interprété comme si nous avions atteint le paradigme idéal. Bien au contraire, ces interfaces possèdent de sérieuses limitations parmi lesquelles la restriction de l'interaction à deux degrés de liberté et la présentation des données uniquement sur deux dimensions. De plus, elles nécessitent souvent de faire correspondre une action bien connue dans le monde 3D vers une du monde 2D. Enfin, parmi les limitations les plus notables, on peut parler de la surcharge de l'écran induite par les nombreuses fenêtres d'applications ouvertes sur un système multitâches.

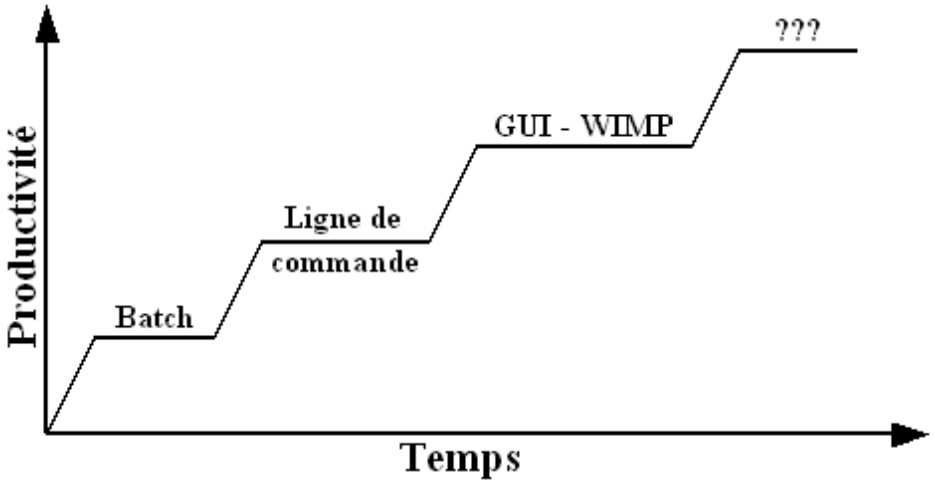


**Fig. 3.1** Les paradigmes d'interface utilisateur.

Malgré leurs limitations, aucun nouveau paradigme n'a encore réussi à faire vaciller les interfaces WIMP (figure 3.1). Pourtant, nous sommes persuadés qu'un nouveau paradigme profitant des nouvelles possibilités 3D peut permettre une plus grande productivité. Utilisées conjointement aux reconnaissances vocale et gestuelle, les technologies interactives 3D comptent en effet bien des avantages sur les interfaces 2D. Sachant que nous évoluons tous dans un monde 3D, nous avons de ce fait une certaine agilité pour manipuler les objets 3D, pour naviguer dans une scène ou encore pour interpréter les relations spatiales. Enfin, les interfaces 3D offrent plus d'occasions d'utiliser notre mémoire spatiale.

Pour ces différents avantages que compte la 3D sur la 2D, il semble qu'elle soit un prétendant intéressant pour remplacer les interfaces WIMP actuelles. Cependant, cela suppose un changement dans son mode d'utilisation actuel. En particulier, il semble fort restrictif

de se cantonner à la réalité virtuelle, sous prétexte de reproduire l'existant, alors que de nouvelles représentations ou interactions peuvent être beaucoup plus efficaces.

Nous étudions dans ce chapitre les différentes alternatives possibles pour remplacer les interfaces WIMP, en particulier les interfaces 2.5D et 3D. Pour chacune de ces deux familles d'interfaces, nous présentons les différents projets reconnus et les avantages qu'elles procurent. Sachant que la création d'interfaces 3D est indissociable des avancées technologiques, nous présentons ensuite les quelques technologies 3D émergentes permettant de construire des scènes interactives complexes.

## 3.2 Interfaces et dimensions

Une interface 3D n'est pas nécessaire pour présenter les informations de manière efficace, bien au contraire. Cependant, puisque nous vivons et percevons les objets en 3D dans le monde physique, l'immersion de l'utilisateur dans une telle interface ne peut que faciliter la perception des choses [Gershon95a]. Néanmoins, on constate qu'à cause du coût supplémentaire causé par les calculs nécessaires à la 3D, la plupart des interfaces innovantes de ces dernières années sont des interfaces zoomables en 2D. Nous les appelons également interfaces 2.5D puisqu'elles autorisent un grossissement des objets graphiques (par zoom ou par loupe) sans être véritablement 3D (pas de rotation du point de vue, par exemple).

Les interfaces dites 2.5D n'effectuent aucune tâche qu'une interface 3D ne pourrait effectuer. Cependant, ce gain de temps, lors des calculs des images de l'interface, permet de garder un taux de rafraîchissement suffisamment élevé pour offrir une animation fluide et pour que l'utilisateur ne perde pas le contexte des informations dans lesquelles il se trouve. Les calculs sont moins lourds car ils prennent uniquement en compte le déplacement et la transformation homothétique de l'image d'un objet en fonction du niveau de zoom et de la position de l'observateur.

Contrairement, pour une interface 3D, le temps nécessaire aux calculs avant l'affichage de l'image résultante ne permet pas, dans certains cas, de garder une animation fluide. La majorité des technologies 3D fonctionnent en effet sur le mode AFAP (*As Fast As Possible* – Aussi vite que possible) ne garantissant aucunement un taux de rafraîchissement constant. L'interaction de l'utilisateur n'est alors pas immédiatement reflétée à l'écran et cet affichage décalé que l'utilisateur perçoit comme des sauts plutôt que comme des déplacements le trouble pour comprendre, après chaque trame, quelles sont sa position et son orientation par rapport aux données. Cette compréhension nécessite un effort cognitif rendant l'interface moins "efficace". Nous voyons ici un aspect important que doivent respecter, dans la mesure du possible, les interfaces. Garder un taux de rafraîchissement interactif autorise en effet l'utilisateur à se reposer sur son système perceptif, sans effort cognitif superflu, pour se repérer.

Toutes les interfaces que nous étudions lient l'interaction de l'utilisateur avec le filtrage des données. Ce filtrage est effectué soit par l'utilisation d'une caméra mobile qui visualise les données, soit par le choix de la donnée à visualiser et par la présentation de son contexte. Toutes permettent, comme nous le verrons, d'éviter la navigation hiérarchique ou séquen-

tielle 2D traditionnelle dans un graphe (à base de liens vers les nœuds fils et le nœud père d'un nœud courant).

### 3.2.1 2D - Interfaces graphiques

Les interfaces en deux dimensions, dites interfaces graphiques, sont encore omniprésentes depuis leur apparition il y a de cela une trentaine d'années. Le système de fenêtrage est la partie de l'interface entre l'homme et la machine qui s'occupe de l'affichage et des dispositifs d'entrée/sortie [Foley90]. Ce gestionnaire est, selon les cas, inclu dans le système d'exploitation comme sous Windows ou externe comme XWindow sous les environnements Unix. Quelle que soit la manière de l'associer au système d'exploitation, il remplit toujours les mêmes tâches :
- gestion et manipulation des fenêtres, des menus, des boîtes de dialogue créés par les différentes applications,
- gestion et répercussion vers l'application adéquate des interactions effectuées à partir de la souris et du clavier,
- détermination de l'apparence (*look & feel*) de l'interface utilisateur.

Ces interfaces 2D ont bénéficié des nombreuses recherches menées sur les systèmes de fenêtrage [Bly86, Morris86, Myers88, Teitelman86]. Au milieu des années 80, les gestionnaires de fenêtres des systèmes Unix et MacOS ont convergé vers la métaphore du bureau gérant des fenêtres recouvrantes. Depuis lors, cette métaphore est présente sur les systèmes que nous manipulons. Seulement de légers changements (voire améliorations) ont été opérés et nous les présentons dans les paragraphes suivants. Le problème majeur de ces interfaces 2D vient du fait que les ordinateurs sont utilisés d'une manière différente d'il y a 15 ans. Avec les très nombreux logiciels disponibles et les services en ligne sur Internet, les utilisateurs gèrent désormais de très nombreuses tâches et doivent basculer de l'une à l'autre très fréquemment [Bannon83, Henderson86].

Lorsqu'on utilise la métaphore du bureau, le changement de tâche ou d'application peut nécessiter de nombreuses opérations : icônifier, ouvrir une application, déplacer ou redimensionner une fenêtre. Lors de certains travaux, l'utilisateur peut également avoir besoin de voir plusieurs applications au même moment [Kandegon97]. Cela requiert encore plusieurs opérations (ouverture, déplacement, redimensionnement de fenêtres voire défilement de leur contenu). Dans ces deux cas d'utilisation très courants, la métaphore du bureau est plutôt inefficace, ce qui provoque une frustration et une perte de temps chez les utilisateurs comme l'ont remarqué [Card96, Henderson86, Kandegon97, Robertson93a, Robertson98]. Pour cela, plusieurs techniques ont été expérimentées pour gérer au mieux le basculement des tâches.

#### Redimensionnement automatique des fenêtres

Le système Andrew [Morris86] propose une organisation automatique des fenêtres sans recouvrement. Les fenêtres y sont redimensionnées afin que lorsque l'une grandit les autres rétrécissent de façon à ce que l'ensemble des fenêtres occupe toujours la totalité de l'écran. Cette technique a été très vite abandonnée car les utilisateurs sont désorientés par de telles

mouvements de fenêtres. La mémoire spatiale est mise à rude épreuve par les mouvements semi-automatiques des fenêtres que l'on ne manipule pas directement.

Les fenêtres élastiques [KANDEGON97] utilisent le même genre d'organisation automatique sans recouvrement. Cependant, la différence avec Andrew vient de l'utilisation de conteneurs. Ces derniers, comparables aux salles que nous présentons ci-dessous, peuvent accueillir différentes fenêtres et permettent ainsi aux utilisateurs de faire un classement des fenêtres qui leur convient. Ce projet aborde le problème de l'affichage simultané de plusieurs environnements contenant de multiples fenêtres. Par ailleurs, il est possible, avec ce système, d'utiliser sa mémoire spatiale pour se souvenir de la position des conteneurs à défaut de celle des fenêtres.

### Bureaux multiples

Les bureaux multiples, très peu étudiés et utilisés[1], ont pour but d'étendre la capacité des interfaces graphiques traditionnelles en permettant de gérer plusieurs environnements concurrents. L'utilisateur pourra organiser et répartir les différentes applications et passer d'un environnement à l'autre à son gré. Cette technique permet ainsi de minimiser les fenêtres occultant d'autres fenêtres puisqu'elles sont réparties sur un plus grand bureau virtuel.

Linux est le système d'exploitation qui a le plus démocratisé les bureaux multiples depuis ses toutes premières versions. XFree86, l'implémentation gratuite d'un serveur respectant le protocole X, permet en effet de gérer plusieurs environnements graphiques. De plus, il rend également possible la gestion de bureaux ayant une résolution supérieure à la résolution de l'écran. Ceci, grâce à un système de balayage initié par la souris lorsque son pointeur se trouve sur le bord de l'écran. Cette technique est toutefois fort peu utilisée, tout comme la barre des tâches apparaissant automatiquement sous Windows, car elle provoque de nombreuses erreurs de manipulation.

### 2D Rooms

Les salles 2D [CARD87, HENDERSON86] sont construites sur le principe des bureaux multiples en ajoutant toutefois des dispositifs tels que la possibilité de partager les mêmes objets entre différentes zones de travail, avoir une vue d'ensemble des zones de travail ou encore charger et sauvegarder des zones de travail. Les salles répondent aux problèmes qu'ont les utilisateurs pour structurer et organiser leur travail en utilisant un bureau classique [BANNON83]. Les applications ou données sont réparties dans les salles et l'utilisateur passe d'une salle à une autre en fonction des tâches qu'il veut effectuer.

Les salles sont en fait une représentation visuelle des tâches dès lors qu'on les utilise correctement. Cette métaphore a été appliquée, entre autres, pour mettre en œuvre des environnements de travail collaboratif [PFISTER98]. Les salles 3D [ROBERTSON93A] sont une extension toute naturelle des salles 2D pour les environnements virtuels 3D et peuvent servir de la même manière à organiser les tâches dans un atelier d'informations.

[1] Il n'y a pas véritablement d'articles traitant spécifiquement des bureaux multiples.

## Emotional Icons

Les icônes émotives (*emotional icons*) [Walker95, Rivera96] sont des objets qui modifient leurs comportements et donc leurs actions en fonction de la présence de l'utilisateur ou d'autres icônes. Le but de ces icônes émotives est de rendre l'utilisation du monde de données plus interactive et plus dynamique. Elles peuvent répondre à la présence ou à la proximité d'un utilisateur dans l'environnement, leur comportement pouvant être dépendant du profil de l'utilisateur ou de ses activités et intérêts courants. Les icônes peuvent, par exemple, avancer vers l'utilisateur ou s'en éloigner, s'agrandir, se rétrécir, s'animer ou changer leur apparence en fonction de la pertinence ou de l'importance des données qu'elles représentent. Les icônes réagissent également à la proximité d'autres icônes : celles portant des informations de nature similaire peuvent s'attirer tandis qu'elles repoussent les icônes charriant des informations de type différent. Les icônes émotives sont idéales pour la création d'un environnement de données vivant. Elles peuvent être utilisées aussi bien en 2D qu'en 2.5D ou qu'en 3D.

## Graphes dynamiques

Les graphes dynamiques se rapprochent des techniques dont nous avons parlé dans la section 2.2.4 page 13 pour la disposition automatique des graphes. Ces techniques conventionnelles de représentation des graphes supposent une fonction ou une routine dont le but est de mettre en œuvre une disposition appropriée sur un graphe donné, tout en essayant de satisfaire un certain nombre de critères esthétiques ou heuristiques. Les graphes dynamiques diffèrent de ces techniques conventionnelles en permettant au graphe lui-même de gérer sa disposition en la modelant comme un système physique initialement instable qui cherche à se retrouver dans un équilibre stable. L'utilisation de l'informatique permet d'aller au-delà des comportements spécifiés d'après des modèles physiques réels. Les comportements peuvent, en effet, prendre en considération et s'adapter à différents critères tels que l'efficacité, la vitesse, l'exactitude ou l'esthétique de la réorganisation du graphe.

La première application de cette méthode pour organiser les graphes a été faite par Eades [Eades84] et a été appelée le *spring-embedder* (le "ressort-inclus"). Ce travail d'Eades est issu d'une technique de placement automatique des composants sur un circuit imprimé [Quinn79] et est à l'origine d'un grand nombre d'améliorations, de modifications et d'algorithmes semblables. Les dispositions de graphe produites en utilisant de telles méthodes sont souvent très satisfaisantes et esthétiquement agréables. Des exemples d'applications de tels algorithmes de placement sur un grand nombre de graphes peuvent être trouvés dans [Fruchterman91]. Les *dynamic diagrams* de Paul Kahn en sont un exemple.

Le système d'Eades est modelé en utilisant un réseau d'anneaux et de ressorts. Les nœuds et les arêtes du graphe représentent respectivement les anneaux et les ressorts. Dans ce modèle, les ressorts n'obéissent pas aux lois habituelles régissant leur comportement. Lorsqu'ils sont étirés au-delà de leur longueur normale, ils exercent une force attractive proportionnelle à la taille de l'étirement. Au contraire, lorsqu'ils sont comprimés au-delà de leur longueur normale, ils exercent une force répulsive, encore proportionnelle à la taille de la compression. Un tel réseau se trouvant initialement dans un état instable cherche

naturellement à atteindre l'équilibre. Une fois que cet état stable est atteint, l'organisation est achevée.

Cette technique est souvent employée pour l'affichage de grandes hyperstructures telles que des graphes d'appel de programme, des graphes de contrôle de flux ou des relations entre documents hypertextes. La technique est facilement portable dans l'espace tridimensionnel et produit des structures de graphe très intéressantes et souvent optimales. Des exemples d'applications pratiques de tels algorithmes dans la visualisation d'informations 3D peuvent être trouvés dans [Chalmers95, Hendley95a, Kings95]. Toutefois, chaque modification du graphe le rend instable, ce qui provoque immédiatement la recherche d'un nouvel état d'équilibre. Ces mouvements soudains des éléments du graphe peuvent troubler l'utilisateur et lui faire perdre les nœuds sur lesquels il travaillait.

### 3.2.2 2.5D - Interfaces zoomables

Comme nous le disions en introduction de ce chapitre, la surface de visualisation réduite des interfaces 2D "classiques" ne suffit plus, dans la plupart des cas, pour soumettre de nombreuses informations aux utilisateurs. Une alternative aux interfaces 3D, en attendant que celles-ci soient accessibles à tous, et suffisamment performantes, sont les interfaces 2.5D. Ces dernières sont en fait un sous-ensemble des interfaces 3D dans lesquelles les possibilités ont été simplifiées pour obtenir une grande fluidité. Plus particulièrement, la navigation a été restreinte aux zooms et aux déplacements dans un plan parallèle aux données. La projection de l'environnement en est ainsi grandement facilitée.

Nous ne présentons pas uniquement les interfaces zoomables mais toutes les interfaces à changement d'échelle. Ces dernières rassemblent bien évidemment les interfaces zoomables mais également les interfaces dans lesquelles le point de vue sur les données est fixe, mais dans lesquelles ce sont les tailles des objets qui sont variables. Ces interfaces comportent une partie centrale réservée aux objets que l'on souhaite voir en détail alors que les parties périphériques sont utilisées pour le contexte, c'est-à-dire pour tous les objets proches jusqu'à une distance donnée.

Dans toutes ces interfaces, les éléments graphiques sont identiques (champs de texte, boutons, menus, ...) mais ils sont inclus dans une surface de visualisation extensible (interfaces zoomables) ou "auto-réorganisable". Ces interfaces permettent toutes d'avoir une vision d'ensemble des données et également de voir une information en détail. Elles se différencient cependant sur les types d'interactions fournies aux utilisateurs et l'on peut dégager trois catégories. Pour chacune d'elles, nous donnons des exemples d'interfaces et une brève explication de leurs caractéristiques et de leurs points forts.

#### Point d'intérêt et contexte

Ces techniques visent à étendre l'espace d'affichage disponible pour les informations. Une partie de cette surface est réservée à l'affichage des détails alors que le reste sert au rendu du contexte. Cette famille d'interfaces est issue des travaux préliminaires de Furnas en 1986

sur son interface *fisheye views* [Furnas86]. Parmi ces interfaces affichant tous les objets qu'ils soient dans le point d'intérêt ou dans le contexte, nous pouvons citer les suivantes :

### *Fish-eye Views*

Le nom donné à ce type particulier de visualisations est issu de l'effet semblable obtenu par une vision à travers un objectif de très grand angle[2]. La lentille de type "œil de poisson" distord la vue de sorte que les objets près du centre de l'objectif soient agrandis considérablement. Ce facteur de grossissement optique baisse rapidement en fonction de l'éloignement des objets par rapport au centre. Cela a comme conséquence directe de montrer, dans le plus grand niveau de détails, les objets qui sont le centre d'attention tandis que les objets sur la périphérie sont observés avec peu de détails. Ceci permet d'étudier les objets d'intérêt en détail tout en maintenant toujours une vue du contexte dans lequel se place les objets observés.

Toutes les interfaces de ce type affichent toutes les informations même si celles du contexte sont affichées si réduites qu'elles en deviennent illisibles ou invisibles. Par ailleurs, cette technique introduit une distorsion des informations comme on peut le voir dans la figure 3.2. Bien que peu pratique, ce principe d'utiliser une distorsion dans les visualisations a tout de même été porté de la 2D vers la 3D dans [Sheelagh97].

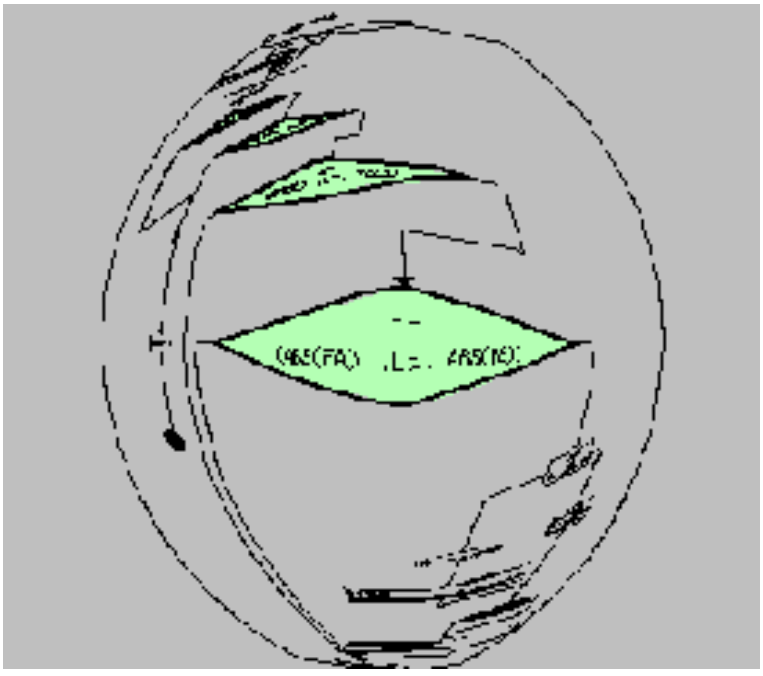

**Fig. 3.2** Une vision de poisson ou *Fish-eye View* [Furnas86].

[2]Ce serait ce que voient les poissons ? !

Les *Fisheye Views* ont été étudiées initialement par G. W. Furnas [Furnas86] mais ont depuis reçu plus d'attention et leur utilisation s'est répandue. Ces vues de Furnas ont été, par la suite, étendues pour autoriser plus de contrôle sur la disposition des objets et pour tenir compte de la structure globale des informations [Sarkar92, Sarkar94]. Elles ont également été employées pour la visualisation de longs menus [Bederson00].

L'utilisation de cette technique de distorsion a pu s'avérer utile pour visualiser de grands graphes. Les nœuds d'intérêt peuvent être amenés plus près du visionneur et, de ce fait, être montrés avec un plus grand niveau de détails tandis que d'autres nœuds sont déplacés automatiquement pour se replacer dans le contexte. Ceci permet à l'utilisateur de se concentrer sur les nœuds intéressants, tout en maintenant toujours une image de la position de ces nœuds dans la structure globale.

### *Les arbres hyperboliques*

L'arbre hyperbolique (*Hyperbolic Tree*) [Lamping95] adapte la technique précédente pour la visualisation de grands arbres de données. Le principe de base de cette technique est la projection de l'arbre sur un plan hyperbolique, puis le plaquage de ce plan comme texture sur un disque pour la visualisation. La construction reprend les principes de la tesselation hyperbolique de Poincaré dont les propriétés suivantes ont attirées l'attention : plus on s'éloigne du centre du disque, plus les éléments sont nombreux sur la circonférence (augmentation exponentielle) et plus leur taille est petite.

L'implémentation de ce principe place initialement la racine de l'arbre au centre, et les nœuds suivants (ainsi que les liaisons entre eux) sont placés sur le disque en fonction de leur niveau de profondeur. Lorsque l'utilisateur clique sur un nœud, celui-ci s'anime et se déplace au centre de manière à ce qu'il soit visible, et l'arbre entier se réorganise en fonction de cette nouvelle racine de visualisation. La projection hyperbolique de cette visualisation permet dans une même aire d'afficher dix fois plus d'informations que dans une surface cartésienne. De plus, l'augmentation exponentielle du nombre de nœuds pouvant être placés sur la circonférence du disque avec l'augmentation de son rayon, correspond au grossissement exponentiel d'un arbre en fonction de sa profondeur.

### *La loupe à documents*

La loupe à document (*Document Lens*) [Robertson93b] est une interface développée au Xerox PARC qui permet de visualiser des documents dont on ne connaît à priori pas la structure (en chapitres, par index, par lien, ...). Cette interface conserve, comme dans l'*Information Visualizer* [Card91], les caractéristiques des interfaces auxquelles croient les membres de Xerox, c'est-à-dire monter des détails d'un objet tout en conservant le contexte (tous les objets).

Son mode de fonctionnement est d'étaler les pages d'un document sur une large surface rectangulaire, et de permettre à l'utilisateur d'afficher les détails d'une partie de la présentation, tout en restant dans le contexte général. Pour cela, l'utilisateur manipule une loupe rectangulaire avec le grossissement voulu et la déplace sur le document pour grossir la partie désirée. Le reste de la présentation (en dehors de la loupe) est recalculé et déplacé

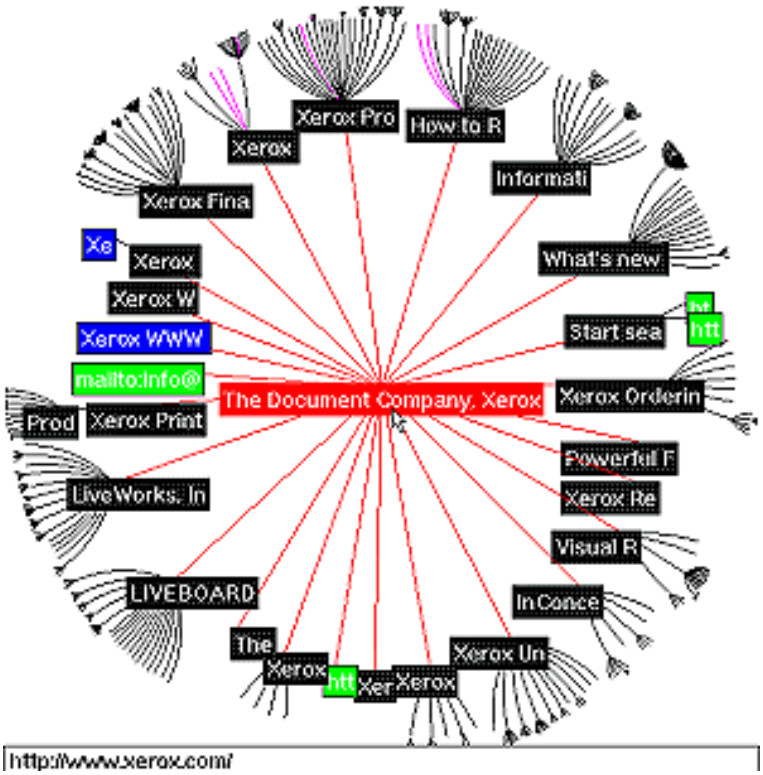


**Fig. 3.3** Un exemple d'arbre hyperbolique [Lamping95].

en dehors de la surface grossie pour que le contexte entier soit gardé. Les transformations affines implémentées pour effectuer ces tâches, permettent de visualiser les documents de manière interactive. Plusieurs essais ont été tentés avec différentes techniques mais chacune a introduit un nouveau problème : dans une interface avec une loupe simple (*magnifying lens*), le contexte en-dessous de la loupe est caché alors qu'avec une interface de type *fisheye view*, la distorsion créée par ce type de loupe rend la lecture d'un texte malaisée. La troisième approche, celle retenue, fut d'utiliser une interface avec une loupe rectangulaire dont les bords "attirent" les éléments voisins, créant ainsi une pyramide à base rectangulaire tronquée. Les bords de la pyramide forment le contexte et sa partie tronquée affiche les détails vus à travers la loupe (comme on peut le voir dans la figure 3.4).

Cette interface répond à des critères devenus essentiels : l'utilisation d'animations fluides, pour que l'utilisateur ne soit pas perturbé par les déplacements de la loupe, et une visualisation de type "point d'intérêt et contexte" pour que l'utilisateur se repère toujours dans la structure globale du document. Ces deux aspects permettent d'utiliser en grande partie le système perceptif de l'utilisateur et moins son système cognitif. Même si l'on ne connaît pas la structure du document manipulé, la conservation du contexte favorise la mémorisation spatiale (on se souvient que la loupe a été déplacée dans une certaine direction par exemple).

**Fig. 3.4** La loupe à documents de Xerox [Robertson93b].

### *Les vues fractales*

Les vues fractales (*fractal views*) [Koike93, Koike95] utilisent les propriétés des arbres fractals pour afficher des structures arborescentes. Leur principe est basé sur l'utilisation de motifs pour la construction des nœuds du graphe. Avec la connaissance de la profondeur N de l'arbre, en appliquant le motif, il est possible de construire la profondeur N+1 (comme schématisé dans la figure 3.5).

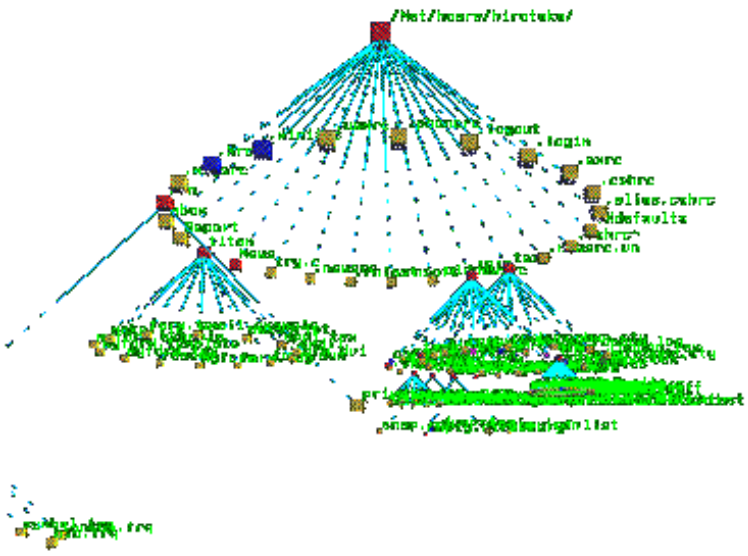

**Fig. 3.5** Une vue fractale [Koike93].

Comme pour l'arbre hyperbolique, ce modèle connaît une augmentation exponentielle du nombre de feuilles en fonction du niveau de profondeur. La partie haute de la figure 3.5 montre un exemple d'arbres fractals.

Toutes les techniques présentées précédemment introduisent une distorsion des données, notamment visible pour les données du contexte. Ces interfaces ne sont donc pas efficaces lorsque le but recherché n'est pas uniquement de percevoir les données, mais plutôt d'appréhender de manière efficace les informations associées.

## Zoom et filtrage

Ces interfaces se distinguent des précédentes par la volonté de réduire l'affichage du contexte par filtrage des données (généralement, le filtrage des données est lié au niveau du zoom). Cette technique est particulièrement utilisée lorsque l'on manipule une grande quantité d'informations, rendant la visualisation globale indésirable voire impossible pour une des raisons suivantes :
- le problème d'occlusion dans le tracé des points d'une information,
- la quantité d'informations est tellement grande que l'on ne peut pas tracer tous les points en un temps raisonnable,
- les données comptent trop de dimensions et il n'est pas pratique de les tracer toutes en une seule fois,
- l'utilisateur est seulement intéressé par la visualisation d'un sous-ensemble des données.

### *PAD++*

La plus connue des interfaces mettant en pratique ce principe est PAD++ [Bederson94] et toutes les applications s'appuyant sur cette interface, comme le navigateur HTML zoomable [Bederson96] de la figure 3.6. PAD++ est une interface qui permet de placer des objets à différentes profondeurs. Les interactions de l'utilisateur, autres que celles avec les objets présents dans l'interface, sont à base de zooms et de déplacements sur le plan (x, y). N'offrant rien de plus qu'une interface 3D, PAD++ est donc typiquement une alternative face à la masse de calculs nécessaires à la 3D.

Dans ce type d'interface zoomable, ce ne sont pas les données qui sont réduites ou agrandies en fonction de la taille du contexte souhaité, mais la caméra (le point de vue sur ces données) qui se déplace pour en faire apparaître plus ou moins. Les données présentes dans l'interface ne sont donc pas distordues puisqu'elles ne subissant qu'un changement d'échelle. La navigation étant contrainte (translations dans un plan horizontal aux données et zooms), l'implémentation des interactions de l'utilisateur est relativement simple à mettre en œuvre[3]. Il est également possible d'implémenter des mécanismes permettant de se rendre directement d'un objet à un autre en suivant un lien. Cependant, l'organisation de l'espace de visualisation est peu facilitée par l'obligation de positionner les éléments sur des plans horizontaux. A cette difficulté, une solution 3D autorisant le zoom, les trans-

[3]Les principes pour implémenter les interactions sont bien décrits dans l'article de messieurs Bederson et Furnas [Furnas95].

**Fig. 3.6** Le navigateur HTML de PAD++ [Bederson96].

lations mais également les rotations, est peut être préférable même si le temps de calcul nécessaire est plus important.

### *Filtering Fish-eye Views*

Les vues de poisson filtrantes (*Filtering Fisheyes Views*) [Sarkar92] sont issues des vues originales de Furnas. Pour une utilisation plus commode d'une telle visualisation, la possibilité de filtrer les données à afficher, a été rajoutée afin de mieux distinguer celles de son choix. Ainsi, la navigation est beaucoup plus aisée puisque les informations du contexte sont beaucoup moins nombreuses, et par conséquent troublent moins l'attention de l'utilisateur.

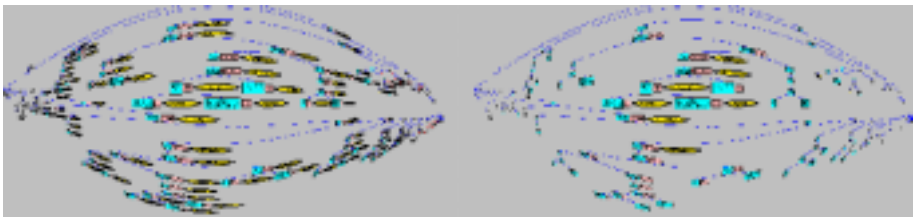

**Fig. 3.7** Un exemple de vue filtrée [Sarkar92].

Une telle interface est pratique pour faire ressortir certaines informations. Par exemple, il est possible d'affiner une recherche dans un corpus, en demandant successivement d'éliminer les livres ne répondant pas à certains critères. Ces vues filtrantes peuvent également être utilisées comme "radar". Elles permettent de filtrer directement, à l'intérieur du radar, les données auxquelles on porte un intérêt particulier, toujours accessibles par un simple clic de souris. Ce principe de filtrage est applicable pour les autres types d'interfaces.

## Widgets de visualisation

Ces méthodes d'interaction permettent à l'utilisateur de sélectionner, par l'intermédiaire de *widgets*[4], l'information qu'il désire voir, de filtrer les informations du contexte, de zoomer parmi celles-ci et de créer des requêtes complexes pour trouver une information particulière. Ce type d'interfaces se distingue des deux précédents par l'utilisation des *widgets* pour la navigation et les opérations sur les objets, au lieu d'une navigation dans l'interface par utilisation directe d'un dispositif de pointage, et des actions par des clics sur les objets. Cette solution a été particulièrement employée pour permettre une navigation sur les six degrés de liberté avec une souris 2D n'en comportant que quatre. La plupart des navigateurs 3D actuels utilisent ce principe de navigation.

### *Les filtres*

Les filtres déplaçables (*Movable Filters*) de Stone *et al.* [Stone94] sont très pratiques pour faire ressortir une information particulière. On peut les voir comme des *widgets* permettant de filtrer localement et rapidement les données. Le principe est d'appliquer une ou des fonctions à un filtre et ensuite de le déplacer sur les données. Le filtre opère alors sa fonction sur les données sur lesquelles il se trouve. Par exemple, dans une base de données de livres, nous pourrions mettre au point un filtre pour afficher les auteurs des ouvrages sur lesquels il est déplacé. La figure 3.8 illustre l'utilisation de filtres montrant les détails sur une carte routière.

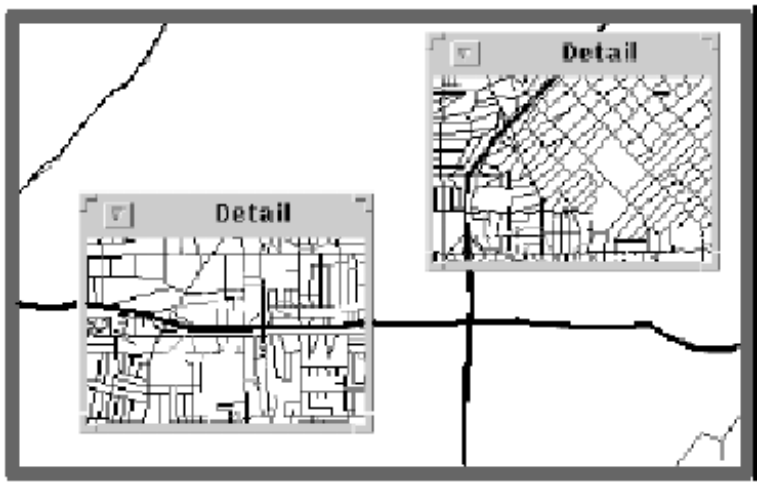


**Fig. 3.8** Un exemple de filtre local [Stone94].

[4] *Widgets* est la contraction de *Window Gadgets*.

### *Les loupes magiques*

Les *widgets* présentés dans [BIER93] sont des éléments transparents, venant en surimpression entre les objets de l'application et l'utilisateur. Ils peuvent être déplacés grâce à un deuxième dispositif de pointage (manipulé par la deuxième main) et leur contenu dépend de l'élément sur lequel ils se trouvent, puisqu'ils renferment les actions possibles sur les objets. Cette manière d'interagir avec les éléments d'une interface permet d'économiser des déplacements intempestifs entre la scène et les *widgets* traditionnels (comme la sélection d'un objet par un clic de souris, puis le déplacement sur un menu ou un bouton pour choisir l'action à exécuter sur cet objet, puis le retour vers la scène).

**FIG. 3.9** Exemple de loupes magiques [BIER93].

Le principe de ces loupes magiques est le même que celui utilisé pour les menus contextuels (c'est-à-dire que les informations et les actions sur un objet s'affichent lorsque l'on clique dessus). Cependant, par rapport aux menus contextuels, les *widgets* dont nous parlons ici peuvent être transparents. Ils permettent, dans ce cas, de voir les informations se trouvant en dessous, contrairement aux filtres présentés dans [STONE94] qui sont opaques et modifient l'apparence des objets sur lesquels ils se trouvent. Dans le cas d'un filtre grossissant, une partie du contexte est cachée comme avec les loupes. Le point le plus notable de ces loupes transparentes est la possibilité de les superposer l'une sur l'autre. Ainsi, en reprenant l'exemple précédent, il est possible d'utiliser une loupe pour grossir les résultats produits par le premier filtre affichant les auteurs d'un livre.

Il faut bien différencier, d'une part, les *widgets* de navigation et d'interaction et, d'autre part, les *widgets* de visualisation comme les loupes magiques. Les premières servent à choisir des mouvements effectués à l'aide d'une souris et les secondes permettent d'ajouter des informations ou de voir une information plus en détail (comme avec l'interface *Document Lens*). Nous verrons le fonctionnement des *widgets* de navigation dans la section suivante traitant de la 3D.

## 3.2.3 Interfaces 3D

Tous les modes de représentation et les techniques de conception basés sur la 3D servent, dans la plupart des applications, à fournir une compréhension accrue d'un modèle comportant en réalité trois dimensions. Leur but principal est de rendre l'environnement de travail, autrement dit l'interface entre l'homme et la machine, plus réaliste et visuel. Pour la simulation et la conception assistée par ordinateur (C.A.O.), les deux premières utili-

sations industrielles de la 3D, le but est de permettre la manipulation et la simulation de comportements selon certains paramètres. Dans le cas de la C.A.O., les pièces mécaniques sont créées, assemblées et testées avant leur mise en fabrication. Les modèles 3D des pièces, associés aux paramètres physiques de simulation (vitesses de rotation, frottements par exemple) permettent d'obtenir les usures au bout d'une durée donnée.

Dans les films, le but recherché par l'utilisation de la 3D, est de créer un maximum d'émotions chez le spectateur en le plongeant directement dans une action irréelle ou difficile à mettre en œuvre par des effets spéciaux physiques ou mécaniques. Dans les jeux vidéos également, la 3D est le support principal pour satisfaire le réalisme visuel et donc l'immersion du joueur. Ce sont d'ailleurs, sans aucun doute, les jeux vidéos qui ont poussé les développements de matériels et de logiciels 3D depuis une décennie, et qui ont permis de voir arriver sur le marché des produits d'entrée de gamme avec de très bonnes performances. Pourtant, la 3D n'a pas qu'un rôle ludique puisqu'elle est utilisée dans les laboratoires de recherche, les centres de conception, pour les études de design, en architecture et également de plus en plus, dans l'art numérique.

Comme on peut le constater en regardant une chronologie des événements ayant marqués l'IHM, un des premiers domaines dans lequel la 3D a été employée est la simulation et la visualisation de phénomènes réels. Dans les laboratoires de chimie, la 3D est utilisée pour obtenir des représentations de molécules complexes manipulables à souhait. La 3D est un outil très puissant permettant la représentation de l'infiniment petit. Même si une molécule peut être perçue au microscope, l'étude de sa structure est rendue bien plus simple par sa représentation 3D. Son utilisation devient obligatoire si l'on étudie des objets encore plus petits, impossibles à voir, même avec les plus puissants microscopes. C'est par exemple le seul moyen de représenter le produit d'une collision entre deux particules à l'intérieur d'un accélérateur.

En astronomie, règne de l'infiniment grand, la 3D est également largement utilisée. Il est possible de suivre le mouvement des astres, ou tout simplement d'obtenir une carte animée du ciel vue d'un point quelconque du temps et de l'espace. On peut ainsi faire des promenades virtuelles dans le système solaire ou dans l'univers. Par simulation, il est possible de visualiser les dégâts occasionnés par une météorite. Une des premières animations 3D, proposée par E. Zajac au BTL (*Bell Telephone Laboratory*), visualisait l'orbite d'un satellite artificiel autour de la terre. A encore plus grande échelle, il est également possible de visualiser les collisions de galaxies.

La 3D est aussi utile en cartographie, pour représenter les informations relevées par les satellites ou par les topographes sur le terrain. A partir des fichiers de données de ces relevés topographiques accessibles à tous, par les fichiers DEM (*Digital Elevation Model*) ou bientôt sur le site Web de l'Institut Géographique National. Grâce à ces modèles de terrain, il est possible de présenter les différentes informations géographiques comme dans le projet de J.M. Le Gallic [Gallic99] ou d'effectuer une simulation dans un milieu en construction [Topol97].

La 3D temps-réel[5] commence également à être employée pour permettre aux utilisateurs d'interagir avec de nombreuses données présentées de façon organisée. Toutes les appli-

[5]On entend par 3D temps-réel, l'utilisation de techniques 3D qui permettent d'obtenir des images de scènes 3D réactives aux interactions de l'utilisateur. A l'heure actuelle, la technique principale pour

cations vues dans le chapitre précédent en sont la preuve. Les interfaces de type WIMP, gérant des fenêtres recouvrantes (*overlapping windows*), montrent leurs limites du point de vue ergonomique. Pour la visualisation, par exemple, il est difficile de soumettre un nombre important de données avec une interface 2D. Nous pensons aussi que le matériel actuel permet d'envisager dans un avenir proche la réalisation d'interfaces graphiques en trois dimensions comme vitrine pour accéder aux applications bureautiques. D'autant plus qu'il apparaît aujourd'hui, que les interfaces 3D sont, au pire, aussi efficaces que les interfaces 2D pour trouver, retrouver et manipuler les données [Sebrechts99]. Toutefois, des études comme celle de Cockburn [Cockburn01] sont arrivées à des résultats différents, révélant toujours un léger avantage à l'interface 2D, mais sans que cet avantage soit rédhibitoire quant à l'utilisation de la 3D. Mais il est à noter, que les expériences menées dans ce dernier article, ne prennent pas en compte l'expérience acquise par les utilisateurs au fur et à mesure qu'ils manipulent l'interface 3D alors qu'ils bénéficient évidemment de l'expérience engrangée avec les interfaces 2D.

Cette section présente une variété de techniques de visualisation d'informations en 3D, largement employées dans beaucoup de systèmes. Ces techniques vont de la présentation usuelle des données par graphiques 3D, jusqu'à la création d'espaces de données abstraits et de leurs comportements à l'intérieur de cet espace. La majorité de ces systèmes de visualisation 3D pourront être classés dans un certain type de groupes. Les graphiques, les "paysages urbains" [6], l'espace de Benedikt et l'arrangement spatial peuvent tous être rangés dans le groupe mettant étroitement en relation (*mappings*) le domaine de données avec celui de l'espace de visualisation. Ces techniques emploient toutes des caractéristiques, propriétés ou valeurs, des données pour projeter les informations dans la visualisation. Les murs en perspective (*perspective walls*), les arbres coniques (*cone trees*), les arbres "penchés" (*cam trees*) et les salles (*rooms*), tous issus des travaux de Xerox au centre de recherche de Palo Alto, peuvent être classés parmi les techniques de présentation de l'information. Ces visualisations se concentrent sur l'apparence, l'accessibilité et l'utilisation des données et ont pour but de fournir une interface facile à utiliser et intuitive. Enfin, les vues d'œil de poisson (*fish-eye views*), les icônes émotives (*emotional icons*) et les graphes auto-organisés, que nous avons vus dans la section 3.2.2 page 67, peuvent tous être décrits comme des techniques de visualisation dynamique. Ces techniques dotent la visualisation de comportements et de propriétés dynamiques, permettant à l'interface de s'adapter automatiquement à des changements des données ou aux actions de l'utilisateur. Ces classifications sont plus ou moins exactes car il peut y avoir un certain degré de chevauchement. Par exemple, les *cone trees* pourraient tout à fait être classés parmi les techniques dynamiques.

## Graphiques 3D

La visualisation 3D la plus naturelle est bien évidemment d'étendre les graphiques 2D standards en 3D. Ces graphiques 3D sont construits tout simplement en traçant un point dans le repère 3D, en utilisant un triplet de valeurs pour chacune des données. En règle

obtenir la cadence d'affichage nécessaire est la projection de facettes. Il ne fait aucun doute cependant que la technique du lancé de rayon deviendra temps-réel à moyen terme.

[6] les *cityscapes* sont issus des *landscapes* avec une apparence de ville, d'où leur nom.

générale, ces triplets se composent de deux valeurs qui ont une structure régulière (par exemple, les jours de la semaine et les heures) et une valeur variable (par exemple, la force du vent ou la température). Les deux ensembles réguliers sont tracés sur le plan horizontal formé par les axes X et Z et les données variables sont tracées comme étant des hauteurs, sur l'axe Y. L'ensemble de points ainsi formé, est utilisé pour former un maillage (*mesh*) ou une surface. Les points sont la plupart du temps également coloriés pour souligner les variations de valeurs. La visualisation résultante ressemble à un paysage qui peut être facilement interprété. Elle permet de repérer rapidement des caractéristiques importantes d'un ensemble de valeurs telles que des répétitions ou des irrégularités.

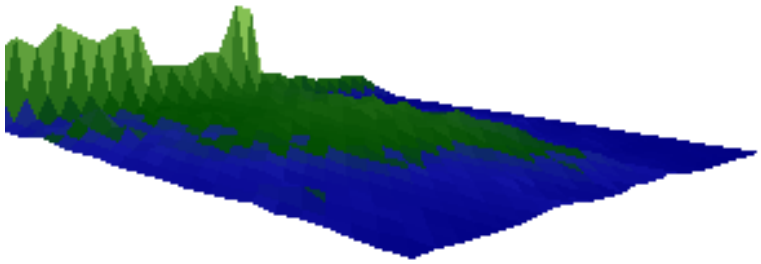

**Fig. 3.10** Un exemple de graphique 3D.

## Cityscapes

Les *cityscapes* (paysages urbains) sont en fait une déclinaison des diagrammes à barres 3D et donc une variation des graphiques 3D. Ils sont créés d'une manière similaire aux graphiques 3D en utilisant des valeurs scalaires pour élever des barres ou blocs 3D placés uniformément sur un plan horizontal 2D. La visualisation en résultant est une représentation plus granulaire de la parcelle de terrain. Une telle visualisation permet à l'utilisateur de se déplacer et de s'orienter plus naturellement, comme dans une ville. Les véritables espaces urbains peuvent même constituer un prototype pour l'implémentation d'environnements virtuels [Ingram96].

Le démonstrateur présenté dans [Walker93], développé pour l'opérateur de téléphonie anglais *British Telecom*, utilise des *cityscapes* pour faciliter la compréhension des graphiques. Certaines fonctionnalités ont été rajoutées. La première est la projection des valeurs minimales, maximales et moyennes de chaque rangée et colonne de la matrice. Une autre fonctionnalité autorise l'ajout d'un calque avec une transparence variable, placé à une hauteur donnée. Ce calque transparent fait ressortir les blocs ou les valeurs de données qui excède la hauteur fixée et permet une identification instantanée et une comparaison entre ces valeurs.

## Espaces de Benedikt

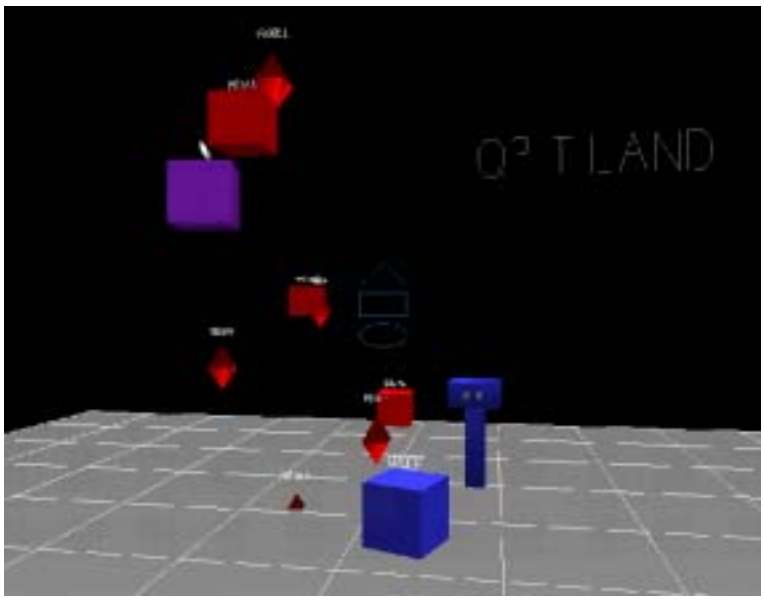

**Fig. 3.11** Un espace de Benedikt [Benedikt91].

L'espace benediktine ou l'espace de Benedikt ou *Benediktine Space* est né des recherches de Michael Benedikt sur la structuration des cyberespaces [Benedikt91]. Benedikt insiste sur le fait que les attributs d'un objet peuvent être tracés sur des dimensions spatiales intrinsèques et extrinsèques. Les dimensions extrinsèques indiquent un point dans l'espace, par exemple un ensemble de coordonnées cartésiennes. Les dimensions intrinsèques spécifient des attributs d'objet tels que sa taille, sa forme, sa couleur, sa texture, etc. Un exemple d'espace de Benedikt pourrait être d'utiliser un attribut tel que des noms d'étudiant sur l'axe des abscisses et leurs notes d'examen sur l'axe des ordonnées. Le diplôme qu'un étudiant recevrait, pourrait alors être tracé sur une dimension intrinsèque telle que l'apparence du marqueur de note.

Benedikt a également proposé deux règles pour son cyberespace, les principes de l'exclusion et de l'exclusion maximale. Ces règles essayent de clarifier le positionnement des données et, en particulier, d'éviter l'encombrement des données et la surcharge de l'interface. Le

principe d'exclusion assure essentiellement que deux données élémentaires ne peuvent exister dans le même endroit de l'espace, c'est-à-dire que leurs dimensions extrinsèques doivent être différentes. Benedikt a énoncé cela comme suit :

> *Two non-identical objects having the same extrinsic dimensions and dimension values, whether at the same time, or including time as an extrinsic dimension from the outset, is forbidden, no matter what other comparisons may be made between their intrinsic dimensions and values*[7].

Le principe d'exclusion maximale étend le principe d'exclusion en assurant que différentes données sont séparées autant que possible, et de ce fait évitant la confusion produite par l'encombrement des objets. Cet aspect est énoncé de la manière suivante :

> *Given any N-dimensional state of a phenomenon, and all the values - actual and possible - on those N-dimensions, choose as extrinsic dimensions - as "space and time" - that set of (two, three, or four) dimensions that will minimise the number of violations of the Principle of Exclusion*[8].

## Perspective Walls

Les murs en perspective [Mackinlay91] s'attachent à la visualisation et au parcours d'informations de grande taille ayant une structure globale séquentielle. Elle permet à l'utilisateur de se concentrer sur un secteur particulier, tout en maintenant toujours son sens de localisation ou de contexte. Les *Perspective Walls* ont une certaine similitude avec les *Fish-eye Views*, vues page 68, puisqu'ils permettent également la visualisation en détail d'un secteur particulier tandis que les données proches de celui-ci sont encore visibles avec un niveau de détail moindre. Tout comme la technique vue précédemment, celle-ci donne donc aussi une idée de la position et de l'orientation dans les données.

Les *Fish-eye Views* et les espaces de Benedikt emploient respectivement les stratégies de partitionnement de l'espace et de partitionnement temporel. Le partitionnement de l'espace se caractérise par des politiques de positionnement (*layouts*) et par des choix de conception graphique pour maximiser la visualisation d'une seule zone tout en présentant le plus d'informations possible. Cette technique souffre de la surcharge d'informations et cela implique qu'avec l'augmentation du nombre de données, il devient de plus en plus difficile d'extraire rapidement les détails. Avec la stratégie temporelle, le principe est de casser la structure globale des informations en un certain nombre de zones distinctes dont seulement une, peut être affichée à un instant donné. Ceci permet au visionneur de commuter entre les différentes zones et de se concentrer sur une information particulière lorsque c'est nécessaire.

[7] Deux objets distincts ayant les mêmes dimensions extrinsèques et valeurs de dimension, que ce soit au même instant ou en incluant le temps comme une dimension extrinsèque dès le début, est interdit, quelles que soient les autres comparaisons pouvant être faites entre leurs dimensions et valeurs intrinsèques.

[8] Étant donnés tout état à N dimensions d'un phénomène et toutes les valeurs - présentes et possibles - sur ces N dimensions, choisissez comme dimensions extrinsèques - comme "espace et temps" - un ensemble de (deux, trois, ou quatre) dimensions qui réduiront au minimum le nombre de violations du principe de l'exclusion.

**Fig. 3.12** Le *Perspective Wall* de Xerox [Mackinlay91].

Le problème majeur de cette stratégie est de se perdre rapidement dans les données car aucune vue n'est fournie pour donner une indication sur le positionnement global.

Les *Perspective Walls* (figure 3.12) tentent de régler les problèmes précédents en étendant efficacement la stratégie temporelle par des indications contextuelles. La structure linéaire y est pliée dans l'espace 3D, par exemple en formant une coquille cylindrique avec les données tracées sur la surface intérieure (mais d'autres configurations sont possibles). Une zone de données peut être visionnée en détail à tout moment, alors que les zones adjacentes sont pliées de chaque côté pour donner des indications sur la position de la zone courante. Pour le déplacement, le mur tourne de manière fluide pour amener la prochaine zone au centre et mettre à jour le contexte.

Les *Perspective Walls*, les *Cones Trees*, *Cam Trees* et les salles 3D, décrits ci-dessous, sont le résultat des recherches sur la visualisation d'informations structurées, effectuées au centre de recherches de Xerox à Palo Alto. Toutes ces visualisations sont intégrées dans un visualiseur général : l'*Information Visualizer* [Card91].

## Cones Trees et Cam Trees

Les arbres coniques (*Cone Trees*) et les arbres "allongés" (*Cam Trees*) [Robertson91] sont des techniques tridimensionnelles pour visualiser les structures arborescentes. La distinction entre les deux types de visualisation se fait sur l'orientation de l'arbre : verticale pour les arbres coniques et horizontale pour les arbres "allongés" (d'où leur nom en français). Le but de ces visualisations d'arbres est de soumettre à l'utilisateur une plus grande quantité d'information, à l'intérieur de laquelle il peut naviguer de manière intuitive. De cette manière, une partie de la charge cognitive nécessaire pour comprendre la structure des informations, est soumise au système perceptif.

Les *Cone Trees* (voir figure 3.13) sont construits en plaçant le nœud racine à l'apex d'un cône translucide près du plus haut point de l'écran. Tous les nœuds enfants sont distribués à distances égales le long de la base du cône. Ce processus est répété pour chaque nœud de la hiérarchie en utilisant à chaque niveau un diamètre de plus en plus petit pour la base

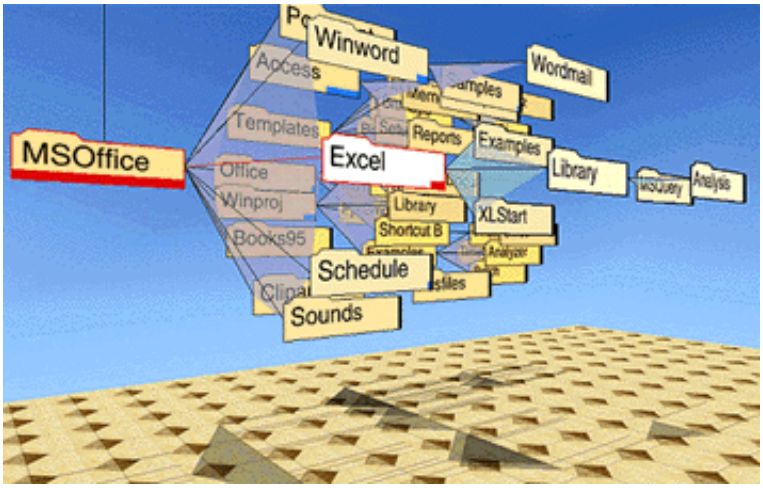


**Fig. 3.13** Le *Cam Tree* de Xerox [Robertson91].

des cônes, afin de finir avec suffisamment d'espace pour disposer tous les nœuds feuilles sans chevauchement.

Les versions originales des arbres coniques et "allongés", produites au Xerox PARC, permettent à l'utilisateur de tourner l'arbre sans à-coup afin de visualiser tout nœud de la hiérarchie. L'animation sans heurt s'est avérée critique pour maintenir le modèle cognitif fait par l'utilisateur de la structure globale. Les changements soudains de l'orientation de l'arbre demanderaient un certain temps à l'utilisateur pour réorienter de la même manière son modèle cognitif.

## Visualisation sphérique

La visualisation par sphère est décrite par Fairchild [Fairchild93] comme étant une version intégralement 3D des murs en perspective. Ce type de visualisation est employée dans le système de visualisation *VizNet* (voir page 33) pour souligner les associations entre des objets multimédias. Chaque objet est disposé sur la surface d'une sphère proche des objets qui lui sont relatifs. Les objets moins pertinents sont affichés plus loin et deviennent ainsi moins visibles au fur et à mesure qu'ils se positionnent de l'autre côté de la sphère. Cette technique fournit une visualisation naturelle de type *fisheye* (voir dans la section 3.2.2 page 68) qui fait ressortir les objets d'intérêt et occulte les objets moins importants.

L'information est présentée sur la surface d'un certain nombre de sphères emboitées les unes dans les autres (comme on peut le voir dans la figure 3.14). Ceci fournit un mécanisme simple pour représenter différents niveaux pour les informations hiérarchisées. L'objet d'intérêt est plaqué sur la sphère extérieure (la plus visible de toute) avec ses objets associés s'étalant tout autour de la surface de sa sphère. Les objets qui sont indirectement liés, sont considérés comme étant des objets de plus bas niveau d'intérêt et sont donc plaqués sur des sphères englobées dans la sphère externe. La couleur des sphères devient plus foncée, en fonction du niveau d'encapsulation, afin de fournir à l'utilisateur des indications visuelles quant au niveau de profondeur courant. La navigation est également facilitée par

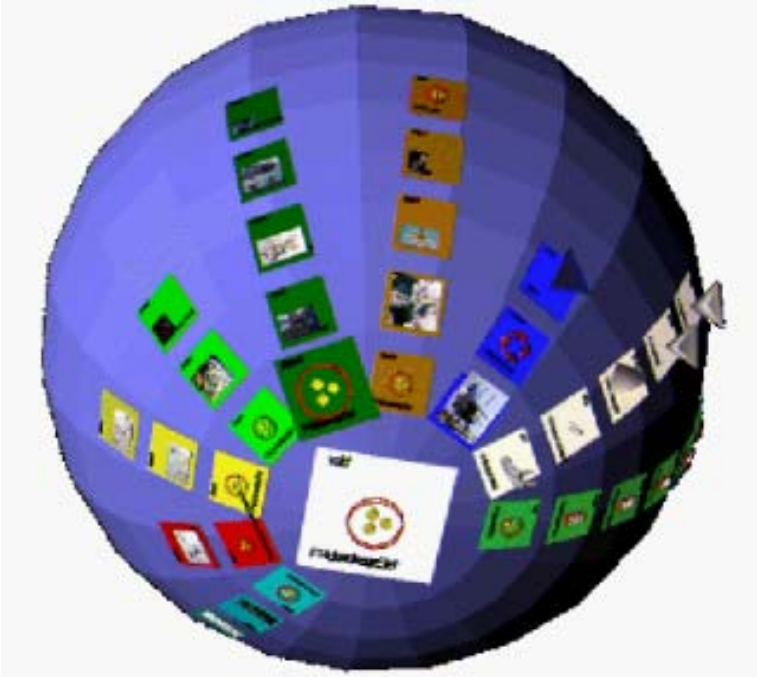

**Fig. 3.14** Visualisation sphérique produite par VizNet [Fairchild93].

des interactions simples permettant de tourner la sphère, pour rendre visibles des nœuds d'intérêt, et de traverser des liens jusqu'aux sphères de plus bas niveau.

## Salles 3D

Les *3D Rooms*, encore développées au Xerox PARC et intégrées à leur outil de visualisation d'informations, l'*Information Visualizer* [Card91], sont une extension 3D du concept de salles 2D, vu dans la section 3.2.1 page 65. Dans chaque pièce, il y aura, en fonction des tâches qui lui auront été assignées, une variété de sources d'informations. Dans l'*Information Visualizer* par exemple, à une salle sera assignée la visualisation de documents séquentiels par le *Perspective Wall*, dans une autre les informations hiérarchisés par les arbres coniques, etc. Les objets 3D présentés dans ces salles peuvent aussi bien être des documents, des données ou des applications. Les murs des salles peuvent également servir de support aux informations, en les utilisant comme des surfaces d'affichage 2D ; comme cela a été fait dans l'environnement collaboratif 3D Spin [Dumas99a].

Les diverses salles, dans un tel "complexe informationnel", sont reliées par un certain nombre de portes. Les portes sont un chemin direct d'une pièce à l'autre. Partir d'une salle par la porte de derrière nous ramènera par exemple dans la salle où nous étions précédemment. Tout comme pour se déplacer dans un bâtiment inconnu, une aide à la visite est nécessaire. D'où la mise à disposition d'une vue d'ensemble sous la forme d'un plan des étages et des salles, y incluant des indications sur le contenu et sur la teneur des tâches associées aux différentes salles. Ceci permet à l'utilisateur de s'orienter mais aussi de se déplacer rapidement entre les salles, et les tâches indépendantes, sans devoir passer par un

certain nombre d'autres salles. Une autre fonctionnalité très métaphorique, incluse dans l'*Information Visualizer*, est la notion de "poche" qui permet à l'utilisateur de transporter certaines informations d'une salle à l'autre et de les garder à portée de main.

## Information Cube

Le cube d'informations (*Information Cube*) est une technique développée par Rekimoto et Green [Rekimoto93] pour visualiser des informations hiérarchiques en utilisant des cubes translucides imbriqués[9]. L'*Information Cube* s'est inspiré des cartographies d'arbre (*tree maps*) de Jonhson et Shneiderman [Johnson91, Johnson92, Shneiderman91]. Dans ces cartes, les informations hiérarchiques sont affichées dans un cadre rectangulaire 2D en essayant de maximiser l'espace d'affichage. Pour cela, le cadre disponible est divisé en un certain nombre de boîtes englobantes qui représentent la structure arborescente. Les éléments parents sont subdivisés en boîtes qui représentent leurs enfants, et ainsi de suite. A chaque information peuvent être associés un degré d'intérêt ou une mesure d'importance qui auront comme conséquence l'attribution d'une plus ou moins grande portion de l'espace d'affichage.

**Fig. 3.15** Une visualisation de type *Information Cube* [Rekimoto93].

[9] Il existe un projet presque équivalent développé par Pearson et Steinmetz [Pearson93].

L'*Information Cube* (figure 3.15) étend efficacement les cartes 2D en trois dimensions. Les cubes translucides emboîtés ont un niveau de transparence variable pour dénoter la "profondeur" des informations. La transparence et le coloriage sont les principales techniques employées pour illustrer le contenu des informations. La transparence permet ainsi à l'utilisateur de voir les informations contenues dans un cube mais également de discerner celles de ses cubes enfants (emboîtés). Cette réduction progressive de la transparence, et donc l'occlusion progressive des informations en fonction du niveau de profondeur, simplifie et éclaircit considérablement l'interface. Sans cela, les informations seraient incompréhensibles. Un titre est attaché à la surface de chaque cube pour pouvoir l'identifier plus rapidement. Les nœuds feuilles, qui ne contiennent pas de cubes emboîtés, sont représentés sous la forme de labels 2D. Les cubes peuvent contenir tout type d'informations et ne doivent pas obligatoirement contenir d'autres cubes. D'autres visualisations 3D ou d'autres informations peuvent également être intégrées dans un cube.

### *Widgets* DE NAVIGATION

Nous avons vu précédemment, qu'il existe des *widgets* de visualisation (voir page 74). Pour la navigation aussi, les *widgets* sont utilisés et ont différents modes de fonctionnement :

- La sélection du type de navigation en cliquant sur un bouton schématisant le mouvement souhaité. Les déplacements de la souris dans la zone de rendu, avec son bouton appuyé, servent à calculer les mouvements 3D correspondants (à gauche dans la figure 3.16).
- Le *widgets* sert directement à effectuer les mouvements. Les mouvements ne se font plus en deux étapes (sélection du type de mouvement puis déplacement dans la zone de rendu) mais directement sur le *widget*. Cependant, il est nécessaire de déplacer la souris dans la zone de rendu pour effectuer une sélection.
- Les *widgets* permettant les déplacements de l'objet (et non plus du point de vue sur la scène) viennent en surimpression lorsque l'objet est sélectionné. La figure 3.16 en est un exemple : on y voit les trois axes (représentés par des segments de droite) du repère local et la boite englobante de l'objet sélectionné. L'objet peut être déplacé par glisser-déposer sur les points schématisant les axes. Il peut également être agrandi ou rétréci en déplaçant les sommets de la boite englobante. Le *widget* donné en exemple à droite dans la figure 3.16 s'applique à la 3D mais des *widgets* équivalents existent en 2D pour la manipulation des objets (comme ceux pour étirer un bloc dans les logiciels de P.A.O.).

### EXEMPLES

Plusieurs exemples d'interfaces 3D pour accéder à des informations ont déjà été décrits dans la section 2.3 traitant des domaines d'applications (à partir de la page 16). Il existe un nombre considérable de projets tels que ceux-ci. Il serait bien trop long de les aborder tous ici. Nous pensons néanmoins, que les plus significatifs ont été vus précédemment.

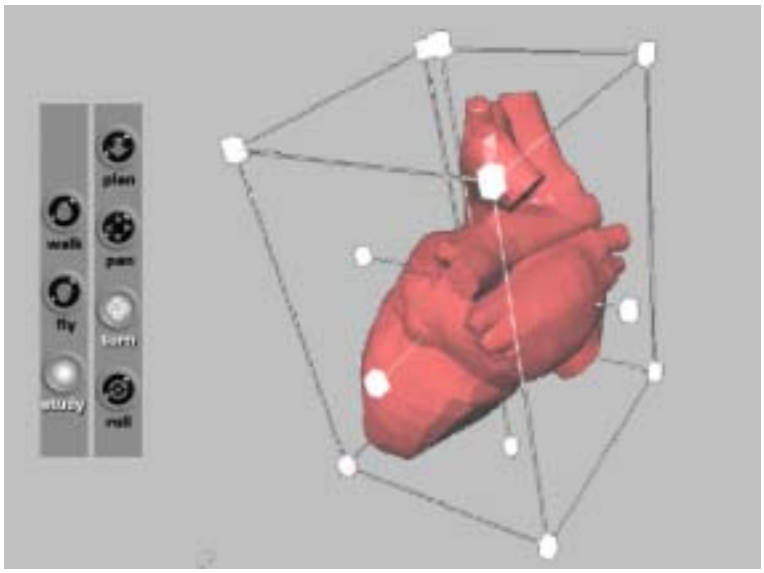


**Fig. 3.16** Exemple de *widgets* de navigation.

## 3.3 Avancées technologiques

Les différents systèmes et techniques de visualisation en trois dimensions n'auraient probablement pas connu un tel engouement, sans les avancées technologiques que nous avons connues ces dernières années. En particulier, les performances offertes par les cartes graphiques et les microprocesseurs d'entrée de gamme démocratisent l'utilisation de la 3D. Il est ainsi possible, désormais, de produire une application qui peut être exécutée sur la majorité des ordinateurs personnels, et non plus uniquement confinée sur des stations de travail spécialisées.

Outre les progrès du matériel pour accélérer l'affichage graphique, par la prise en charge des fonctions 2D et 3D, d'autres aspects ont également connus, de manière complètement autonome ou non, de spectaculaires expansions. Les logiciels 3D ont, en effet, suivi le mouvement et aujourd'hui, les bibliothèques de fonctions 3D, les modeleurs et moteurs 3D se sont multipliés.

Non seulement les matériels et les logiciels graphiques ont évolué mais aussi les technologies permettant à l'utilisateur d'interagir avec les scènes 3D présentées. Dès lors que la 3D par projection de facettes a permis d'atteindre une rapidité telle qu'on l'a qualifiée de 3D temps-réel, les recherches sur les outils d'interaction ont pris une grande importance. Que ce soit pour la conception et l'étude de dispositifs de pointage mieux adaptés à la 3D, ou pour l'étude des mécanismes logiciels d'interaction, on note une évolution nette du domaine.

Parallèlement à cette progression de l'infographie 3D et de ses domaines connexes, d'autres aspects du monde multimédia se sont développés. Notamment, nous voyons l'apparition de bibliothèques de fonctions sonores 3D comme EAX ou OpenAL qui préfigurent la même progression en son que celle entamée en infographie 3D, il y a quelques années. Cet aspect

sonore nous intéresse particulièrement parce qu'il est le complément idéal des graphismes 3D, pour étendre l'expérience immersive.

### 3.3.1 Architecture des systèmes 3D

Le processus menant de la description d'une scène (par un langage prévu à cet effet ou par une série de commandes d'une bibliothèque 3D[10]) à son affichage à l'écran est complexe. Le pipeline 3D, décrit dans [Foley90, Watt93] ou tout autre livre traitant de la 3D, effectue la série d'opérations nécessaires pour afficher une scène. Cependant, il n'effectue pas tous les calculs et ne fournit pas toutes les ressources utilisées pour l'obtention d'un résultat visuel. En aucun cas, en regardant un pipeline 3D, on ne peut expliquer comment des scènes conséquentes sont affichées à une fréquence de rafraîchissement telle qu'elle permet une interactivité. Pour cela, nous devons étudier l'ensemble du processus de production qui passe par différentes phases, chacune d'elles prise étant en charge par une couche spécifique. L'ensemble de ces couches forment une architecture 3D complète qui est schématisée dans la figure 3.17.

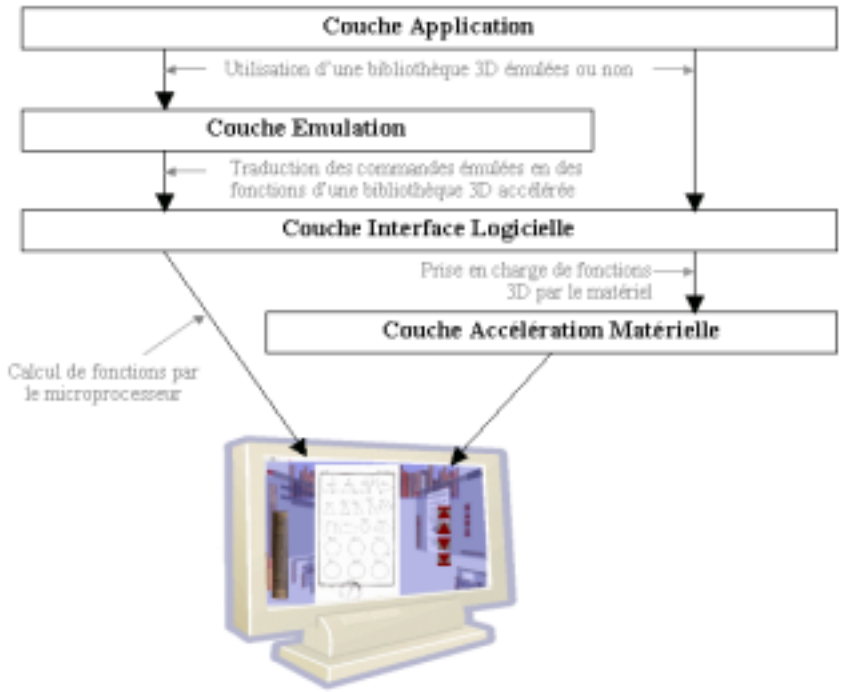


**Fig. 3.17** Les différentes couches d'une architecture 3D.

[10] Ce que nous appelons, pour les distinguer, la 3D décrite et la 3D programmée.

## Couche Accélération Matérielle

La carte graphique, composant essentiel de cette couche accélératrice, a pour but premier de gérer l'affichage des données à l'écran. Depuis vingt ans, plusieurs normes graphiques se sont succédées : CGA, EGA, VGA, SVGA et XGA. Grâce à cette progression, les cartes vidéos ont pu offrir une définition de plus en plus fine et un choix de couleurs de plus en plus important. En général, une carte vidéo est composée des éléments suivants :

- un processeur graphique (GPU - *Graphic Processir Unit*), qui exécute rapidement diverses opérations graphiques (rotation, translation, coloriage, remplissage...) à la demande du microprocesseur,
- une zone de mémoire vidéo à accès direct et dédiée à l'image affichée à l'écran (composée de puces de mémoire dont le niveau de performance varie : DRAM, EDO DRAM, VRAM, SDRAM, SGRAM),
- un contrôleur graphique, qui reçoit les informations numériques en provenance du processeur central et les transforme en une matrice de points qu'il écrit dans la zone de mémoire dédiée à l'affichage. Un convertisseur numérique-analogique (ou RAMDAC, pour *RAM Digital-Analog Converter*), lui permet de convertir la matrice de points en signaux analogiques qui vont piloter le canon à électrons du moniteur lors de l'affichage.

Les cartes 2D utilisent le même principe depuis leur création. Chaque processeur possède de nombreux circuits câblés contenant toutes les informations pour réaliser la plupart des fonctions simples, telles que le déplacement des blocs ou du curseur de la souris, le tracé des lignes et des polygones plans ainsi que leur remplissage, etc. Maintenant que toutes les cartes gèrent ce genre de fonctions et disposent d'une mémoire suffisante pour atteindre des résolutions importantes, on assiste à une stagnation des performances 2D. Désormais, ces performances dépendent uniquement de l'architecture des cartes ainsi que de la qualité de leur *driver*. Le type de mémoire embarquée a également une importance majeure pour la vitesse d'affichage. Pour exemple, l'emploi de mémoire EDO à 60 ns plutôt que des SGRAM ou WRAM (mémoires vidéo spécifiques) à 10 ns peut faire chuter les performances d'une façon étonnante. Enfin, la capacité de la mémoire vidéo et la fréquence du RAMDAC ne permettent en aucune façon l'accélération de l'affichage 2D, mais autorisent simplement un plus grand confort, grâce à une palette de couleurs plus importante ainsi qu'un taux de rafraîchissement élevé, même dans les hautes définitions.

Avec l'arrivée de nombreux logiciels et jeux vidéos ayant recours aux graphiques en 3D, des circuits câblés spécifiques ont été ajoutés aux fonctions 2D déjà présentes. C'est en 1995 qu'apparaît le premier accélérateur 3D grand public, disponible sous la forme d'une carte fille reliée à la carte vidéo principale. Par la suite, les accélérateurs 3D ont pris place directement sur la carte vidéo principale. Aujourd'hui, la quasi totalité des cartes graphiques disposent d'un accélérateur 3D afin de décharger le processeur des tâches affectées au rendu. L'accélération 2D n'étant plus un enjeu majeur pour les fondeurs de cartes graphiques, ce sont les performances des fonctions 3D qui ne cessent d'être améliorées.

Le nombre et le type de fonctions 3D accélérées dépendent de la carte graphique utilisée. Comme cela est illustré dans la figure 3.18, toutes les générations de cartes graphiques n'accélèrent pas les mêmes étapes du pipeline 3D. Comme nous pouvons le voir, et cela rejoint notre propos du paragraphe précédent, les cartes 3D ne cessent d'étendre, depuis leur apparition jusqu'à aujourd'hui, leur prise en charge des calculs nécessaires à la 3D.

Les cartes 2D ne gèrent aucun aspect propre à la 3D. Les dernières cartes uniquement 2D du marché permettent cependant le transfert rapide de zone mémoires. Ainsi, il est possible d'effectuer des animations fluides grâce à une gestion du *double-buffer*. Avec une prise en charge totale des calculs 3D par le microprocesseur, ces cartes suffisent pour afficher des scènes simples avec un mode de coloriage peu coûteux.

En plus des zones mémoires où sont stockées les images à afficher à l'écran, la première génération de cartes 3D prend en charge les calculs de rendu des scènes 3D. Une zone mémoire supplémentaire est ajoutée à la carte graphique pour y stocker les textures. Depuis lors, l'utilisation de la mémoire embarquée sur les cartes 3D a toujours été la même : une moitié de la mémoire pour l'affichage et l'autre moitié pour les textures utilisées lors du rendu. Les premières fonctions 3D câblées ont été le plaquage de texture et la gestion du *Z-Buffer*. Les cartes 3D de deuxième génération ont étendu les possibilités des cartes précédentes en offrant des fonctions plus avancées pour le rendu des objets telles que l'antialiasing des scènes, le filtrage bi et tri linéaire. La partie supérieure du pipeline 3D (transformations, *clipping*, tests de visibilité, gestion des sources lumineuses) est cependant toujours prise en charge par le micro-processeur.

La troisième génération de cartes 3D intègre un nouveau module que l'on nomme *Transform and Ligthning* (T&L ou TnL). Ces nouvelles fonctions permettent à une telle carte d'effectuer les projections parallèles et orthogonales, les transformations géométriques, le *clipping* et les éclairages et ainsi de traiter seule le processus entier de conversion des triangles en des pixels d'une image.

Le microprocesseur ne gère plus que les aspects inhérents aux mouvements de l'utilisateur, c'est-à-dire du point de vue sur la scène, et aux mouvements des objets. Dans une scène animée et/ou hautement interactive, cette part du travail peut toutefois représenter 50% du temps de calcul total nécessaire. Sachant qu'une carte 3D de troisième génération effectue les opérations du pipeline graphique, au pire en deux fois moins de temps qu'un microprocesseur récent alors l'utilisation d'une telle carte offre au moins un gain de 25% par rapport à l'utilisation unique d'un microprocesseur.

Depuis l'apparition du Pentium MMX (*Matrix Math eXtension*) d'Intel, les microprocesseurs disposent également d'instructions spécialisées pour les applications multimédias. Avec les instructions MMX, aucune accélération 3D n'était cependant possible car les 57 nouvelles instructions ne permettaient que d'effectuer des opérations sur les entiers, alors que les calculs pour la 3D sont essentiellement sur des flottants. Pour combler cette lacune, AMD a étendu le jeu d'instructions MMX par 21 instructions supplémentaires permettant des opérations sur les flottants. Ces nouvelles instructions, baptisées 3DNow !, permettent d'accélérer notablement les applications 3D. Devant ce succès, Intel a emboîté immédiatement le pas à AMD en proposant les instructions SSE (*Streaming SIMD Extensions*) puis SSE2 pour prendre en charge les applications nécessitant de nombreux calculs sur les flottants. Une synthèse très complète a été effectuée sur ce sujet par H. Doornbos pour son oral probatoire du CNAM [Doornbos00].

On peut se demander, après avoir vu la figure 3.18, si les instructions spécialisées des microprocesseurs servent encore à quelque chose, puisque les nouvelles cartes accélèrent la majorité des calculs nécessaires à l'affichage d'une scène 3D. Or, les cartes 3D ne s'occupent que de l'affichage et ne prennent absolument pas en charge les à-côtés. Ainsi, les

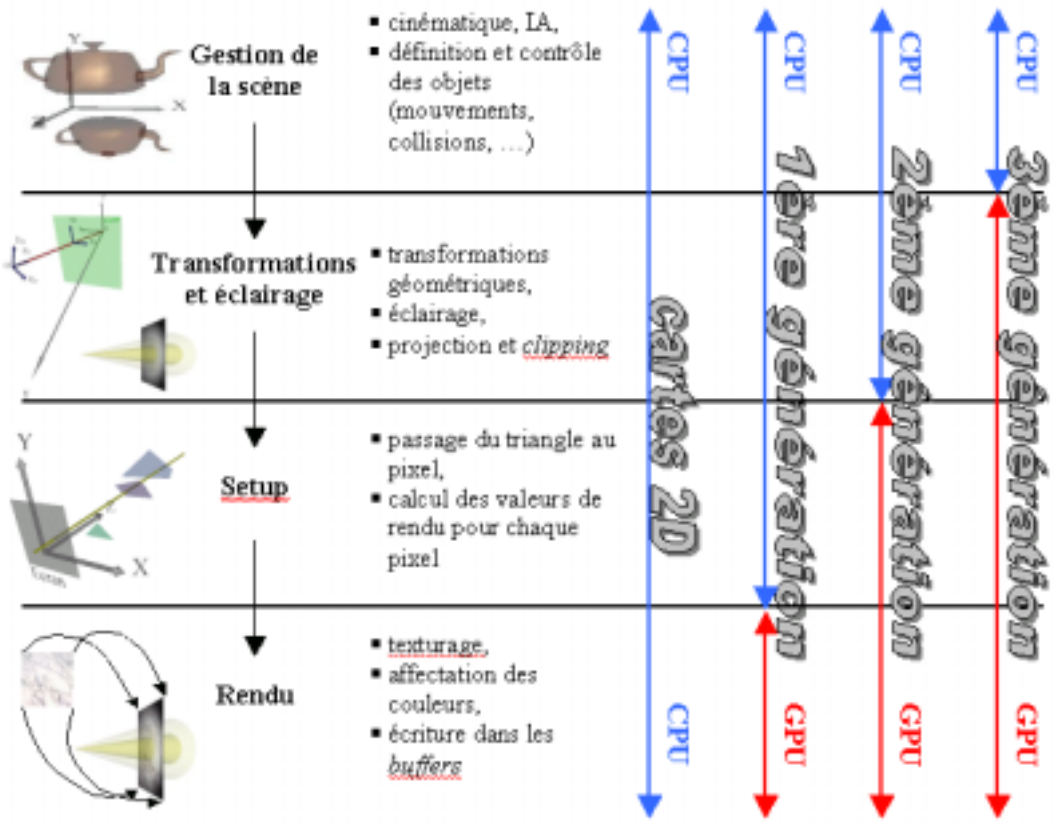


**Fig. 3.18** Fonctions accélérées par les trois générations de cartes graphiques 3D.

interactions, les mouvements, les collisions et les autres aspects qui font qu'une application est interactive, devront être pris en charge par le microprocesseur. Il suffit de se rappeler que ces calculs peuvent représenter 50% des traitements dans certaines applications 3D pour se convaincre de l'apport de ces nouvelles instructions spécialisées.

## Couche Interface Logicielle

Dans cette couche logicielle sont regroupées les bibliothèques de fonctions rendant l'accès à une accélération 3D plus aisé. Sans ces API, il serait néanmoins possible d'afficher un contenu 3D mais toutes les opérations du pipeline 3D devraient être explicitement programmées. Lorsqu'on programme une application en s'appuyant sur une API 3D, il suffit de donner une série de commandes (définies par l'API) permettant de décrire la scène que l'on souhaite afficher. Ces commandes définissent la forme géométrique et l'apparence des objets manipulés et les options de rendu. Les différentes étapes du pipeline sont alors entièrement prises en charge, et de manière transparente, par l'API lorsqu'on lui demande d'afficher la scène. En aucune façon le développeur d'une application 3D n'aura besoin de se soucier de la programmation explicite du matériel présent. C'est là que la couche logicielle revêt toute son importance. En effet, s'il fallait programmer les registres d'une carte

3D pour accéder aux fonctions accélérées, il serait évident que la plupart des applications ne seraient vendues que pour quelques configurations précises.

Les cartes 3D sont fournies avec leurs drivers. Ceux-ci se présentent sous la forme de librairies dynamiques effectuant le lien entre les commandes des API 3D et les fonctions câblées sur la carte. Les API supportées sont généralement OpenGL de Silicon Graphics et Direct3D de Microsoft. Alors que pour Direct3D toutes les fonctions sont généralement supportées, pour OpenGL ce n'est que rarement le cas. Les cartes d'entrée de gamme ne proposent en effet qu'une accélération partielle. Deux options s'offrent alors aux fabricants de cartes 3D :

- Écrire un ICD (*Installable Client Driver*) : une implémentation complète (TnL + Rasterization). Le fabricant de carte 3D peut optimiser le driver pour chaque type de CPU (SSE, SSE2 et 3DNow !) afin d'augmenter les performances. Dans un ICD, toutes les fonctions OpenGL doivent être implémentées. Si l'une d'entre elles ne peut pas être effectuée par la carte, le driver passe alors en mode logiciel : la fonction est opérée par le micro-processeur.
- Écrire un MCD (*Mini Client Driver*) : une version allégée de l'ICD dans laquelle n'est développée que l'étape de rasterization (Setup+Rendu).

Quelque soit l'API utilisée, cette couche logicielle doit pouvoir déterminer, si une opération peut être effectuée par le matériel ou si elle doit être émulée par le microprocesseur. Pour cela, toute API dispose d'un mécanisme permettant de questionner le matériel sur ses possibilités. En fonction des résultats, elle aiguillera l'exécution des commandes vers l'un ou l'autre des composants.

Deux familles d'éléments composent donc cette couche logicielle : les API 3D (elles-mêmes constituées d'une librairie et d'un fichier de déclaration de leurs fonctions pour être utilisées depuis un langage de programmation) et les drivers des cartes graphiques (qui effectuent le lien entre les fonctions de l'API et la carte graphique). Le processus menant d'un code source à une application utilisant une API 3D est schématisé dans la figure 3.19.

### Couche Emulation

La couche émulation est présente sur certaines architectures lorsque l'API 3D utilisée par le programmeur est traduite en une autre API. C'est le cas notamment de Java3D et de Fahrenheit dont les différents éléments de leur graphe de scène sont traduits soit en Direct3D soit en OpenGL. De cette manière, ces graphes de scène dans lesquels sont stockées toutes les commandes 3D, bénéficient d'un rendu accéléré par l'intermédiaire de deux autres API. Sans cette traduction, ces API ne sont pas accélérées puisque les fonctions 3D des cartes graphiques ne sont pas câblées pour elles.

On différencie ici les API 3D qui sont traduites en une autre API 3D, en vue d'être accélérée et les applications qui utilisent directement une API. Tous les *bindings* permettant d'accéder aux fonctions OpenGL depuis un langage de programmation (autre que ceux accédant directement aux librairies dynamiques d'OpenGL), peuvent être considérés comme étant des API 3D "émulées". En effet, le programmeur accèdera aux fonctions d'un *binding* qui lui-même accèdera aux véritables fonctions OpenGL. Etant donnée cette définition d'un *binding* OpenGL, on considèrera donc que Java3D en est un. Pour le langage Java, il en

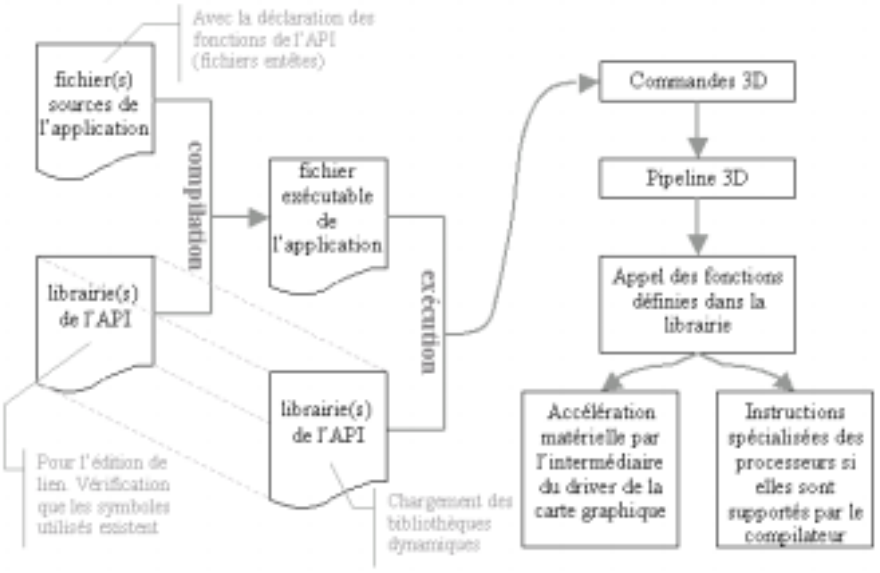


**Fig. 3.19** Processus de création et d'exécution d'une applications 3D.

existe de nombreux autres comme : GL4Java, Magician, YAJOGLB (Yet Another Java OpenGL Binding), Jogl, Tree (interface Java pour Mesa 3D), JavaOpenGL. Mais le langage Java n'est pas le seul à bénéficier des bindings OpenGL puisque l'on peut également en trouver pour les langages Perl, Fortran, Ada et Python.

Les librairies de fonctions qui n'émulent pas entièrement l'API cible, peuvent également être considérées comme des API "émulées". Les librairies d'utilitaires pour OpenGL par exemple, qui regroupent des fonctions permettant de créer des objets complexes, s'appuient sur OpenGL pour construire ces objets. Ces différentes librairies se placent donc bien entre l'application et l'API 3D accélérée par la carte graphique.

## Couche Application

Dans la couche application de notre architecture 3D, nous regroupons tous les logiciels dans lesquels une interface est implémentée à l'aide d'une API 3D. Toutes les applications n'utilisent pas nécessairement une API 3D (accélérée ou non, émulée ou non). Les opérations 3D peuvent, en effet, être calculées par l'application elle-même et ne nécessitent alors que des primitives mathématiques et 2D (pour tracer les points). Cependant, dans la majorité des cas, et par soucis de performance, la quasi-totalité des applications actuelles utilisent un rendu accéléré et c'est pourquoi nous ne considérons que celles-ci dans notre architecture.

Les différents moteurs 3D du Web que nous verrons en détail dans la section 3.3.2 page 93, font partie de cette couche application puisqu'ils utilisent généralement les API OpenGL ou/et Direct3D.

C'est au niveau de l'application qu'un développeur doit interroger une API pour connaître les options de rendu qui sont accélérées. De cette manière, il sera possible de ne pas utiliser certaines options afin de favoriser les performances et, au contraire, d'enrichir la scène si toutes les fonctions sont accélérées.

### 3.3.2 Normes et standards 3D

Il existe deux manières de créer une application 3D. La première consiste à utiliser un logiciel capable de lire une description de scène. La deuxième est de programmer le moteur 3D, en utilisant ou non une bibliothèque de fonctions 3D pour accéder à l'accélération graphique. Les deux solutions, 3D décrite et 3D programmée, ont leurs avantages et inconvénients. Pour chacune d'elles, nous présentons ici les solutions qui tirent leur épingle du jeu.

Les API que nous présentons sont les plus utilisées par les applications exploitant une interface 3D temps réel qui répondent instantanément aux interactions de l'utilisateur. L'association d'un fichier décrivant une scène et d'un moteur permettant de la charger et d'y évoluer peut dans certains cas être insuffisante. En effet, certaines fonctions peuvent manquer ou, au contraire, être trop générales, bref, ne pas être satisfaisantes. En particulier, les différentes interactions gérées par le navigateur ne sont pas nécessairement les plus appropriées pour certaines tâches. La seule alternative pour créer un moteur 3D "sur mesure" est de programmer soi-même ces différents comportements. Cependant, les fonctions de projection, de *Z-Buffering* ou de plaqué de textures, par exemple, restent les mêmes quelles que soient les fonctionnalités voulues. C'est pourquoi, il est fort intéressant, pour économiser un temps précieux dans le développement d'un moteur et pour tirer partie de l'accélération matérielle du rendu, d'utiliser une API 3D regroupant toutes ces opérations élémentaires en 3D.

Nous présentons également à partir de la page 105 quelques logiciels offrant la possibilité de créer et/ou de visualiser des scènes 3D. Ceux-ci se sont multipliés avec l'explosion des performances 3D des micro-ordinateurs et ne sont plus uniquement réservés aux professionnels depuis que leur prix a considérablement diminué. Ces logiciels existent désormais aussi bien sous forme de programmes libres de droits d'utilisation que payants et se regroupent à présent en deux familles principalement :
- les modeleurs qui autorisent la création et l'édition des scènes,
- les moteurs 3D dont le rôle est le rendu des scènes. Ils se différencient des fenêtres de rendu en temps réel des modeleurs les plus récents, en proposant, d'une part, le meilleur compromis possible entre affichage de haute qualité et vitesse de rafraîchissement et, d'autre part, la gestion de techniques d'interaction plus avancées.

Dans ce rapport de thèse, nous ne couvrirons que les moteurs 3D. Ils représentent la manière la plus simple de créer rapidement des scènes 3D interactives, sans avoir besoin de connaissances informatiques autres que celles d'un utilisateur de niveau intermédiaire ou expert, selon les cas. Les modeleurs pourront servir, même si ce n'était pas leur raison d'être au départ, à générer des fichiers décrivant une scène dont le format fait partie des filtres d'import de moteurs 3D. Ainsi, ces derniers seront-ils capables d'interpréter les données

des fichiers et par là même de présenter la scène à l'utilisateur qui pourra interagir avec elle.

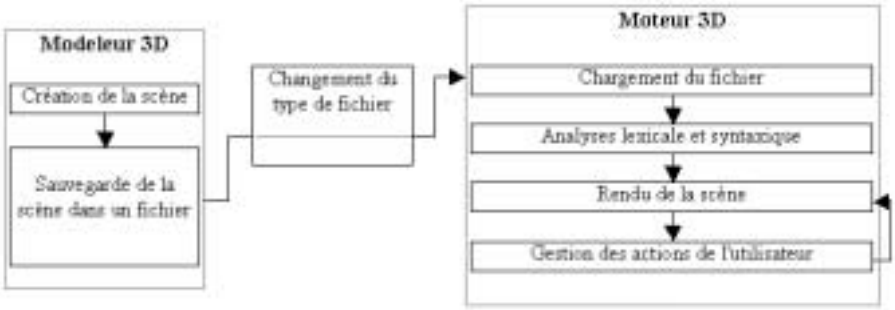


**Fig. 3.20** Chaînes de production et de visualisation d'une scène 3D.

## Les API 3D

Pour toutes les API 3D suivantes, l'intégration de données autres que 3D est à la charge du programmeur. Ainsi, contrairement aux outils 3D présentés à partir de la page 105, les textures animées (vidéos) devront être gérées par l'application. Pour le son, une API adéquate devra également être utilisée.

### *OpenGL*

Le système graphique OpenGL, de Silicon Graphics Inc. (SGI), s'est imposé comme la bibliothèque de référence pour réaliser des projets de synthèse d'images en temps réel, dans le milieu professionnel. Face à son grand succès sur les stations Silicon Graphics, plate-forme sur laquelle elle succède à la bibliothèque IRIS-GL, cette librairie s'est maintenant généralisée sur de nombreux systèmes d'exploitation. Des clones sont d'ailleurs disponibles sur certains systèmes d'exploitation avant que l'originale ne soit portée. Le plus connu de ces clones est Mesa de Brian Paul, disponible pour Linux, Windows et plus récemment sur MacOS.

S'appuyant sur son expérience des bibliothèques de fonctions 3D, SGI a voulu proposer un produit facile à comprendre et à utiliser. Par rapport à IRIS-GL, leur ancienne API, de nettes améliorations ont été apportées :
- Elle est multi-plateforme par sa conception.
- Inspirée par l'interface graphique X Window, elle propose un modèle client-serveur sur réseau hétérogène où le serveur effectue les calculs et le client, l'affichage. Ainsi, le serveur peut être un supercalculateur qui exécute des calculs complexes de simulation alors que le client est un poste de travail principalement utilisé pour ses capacités de visualisation. Cependant, il est possible que tout l'ensemble, gestion du pipeline 3D et affichage, soit traité sur la même machine.

En plus d'être multi-plateforme, la librairie OpenGL peut être utilisée depuis d'autres langages de programmation que le langage C : Fortran, Ada, C++ et Java. Cette caractéristique fait d'OpenGL la librairie graphique utilisable sur le plus grand nombre de plate-formes différentes.

Comparée à Direct3D, OpenGL n'est pas une API de haut niveau dans le sens où elle ne fournit pas de fonctions pour effectuer des tâches complexes comme l'édition d'objet ou la gestion de fichier qui sont laissées au soin du programmeur. Pour manipuler une API de haut niveau exploitant OpenGL, le meilleur choix est d'écrire une application utilisant Silicon Graphics Open Inventor. Cette API propose, en effet, une architecture 3D orientée objet donnant accès à des fonctions de haut niveau. Elle est, pour cette raison, fortement conseillée pour les novices en programmation 3D.

Un point très important dans le développement de cette bibliothèque, est que les cartes graphiques avec fonctions 3D câblées prennent désormais en charge, au niveau matériel, les quelques 250 commandes OpenGL. Jusqu'à récemment, de telles cartes coûtaient très cher et étaient uniquement disponibles sur certaines stations de travail UNIX dont les Silicon Graphics. Actuellement, grâce à la volonté d'ouverture de SGI, toutes les cartes d'entrée de gamme supportent désormais en plus des commandes Direct3D, certaines d'OpenGL (les cartes accélérant toutes les commandes OpenGL demeurent à un prix dissuasif). L'effort des constructeurs va même jusqu'à proposer des drivers pour Linux, preuve qu'ils croient à la fois au développement d'OpenGL et de Linux. Pour les cartes à base de processeurs graphiques Voodoo de 3Dfx Interactive, première famille de carte à avoir proposée un driver sous Linux, cette accélération se fait par l'intermédiaire de la librairie Glide qui sert d'interface avec les registres des cartes.

La figure 3.21 nous montre, d'une part, l'architecture logicielle d'OpenGL et, d'autre part, l'ordre des opérations dans son pipeline graphique. Puisque l'architecture d'OpenGL est de type client-serveur, les calculs et l'affichage de la scène pourront être soit faits sur une même machine, soit répartis selon les possibilités des machines. Dans ce second cas, le pipeline graphique sera géré par le serveur qui recevra les données du client et lui renverra la scène calculée. C'est dans le pipeline graphique que le système détermine si une commande est accélérée par le matériel. Selon le cas, l'exécution de cette commande sera aiguillée vers un module matériel ou un module d'émulation logicielle.

OpenGL compte deux modes de rendu différents : un mode immédiat où les commandes sont envoyées une à une au serveur qui les exécute le plus rapidement possible, et un mode retenu où les commandes sont stockées dans une liste et envoyées en bloc au serveur. Ce dernier mode compte deux avantages importants :
- pour afficher un objet complexe, il suffit de référencer la liste au lieu de redonner toutes les commandes,
- les informations sur un objet sont ainsi envoyées rapidement sur le réseau.

Le seul inconvénient de ce mode retenu apparaît lorsque l'on souhaite modifier l'objet. Il faudra, pour cela, supprimer la liste actuelle et en créer une autre contenant l'objet modifié. Avec OpenGL, il est cependant possible de mélanger des commandes immédiates et des exécutions de liste de commandes, rendant ainsi possible la gestion des objets animés en mode immédiat et ceux fixes en mode retenu.

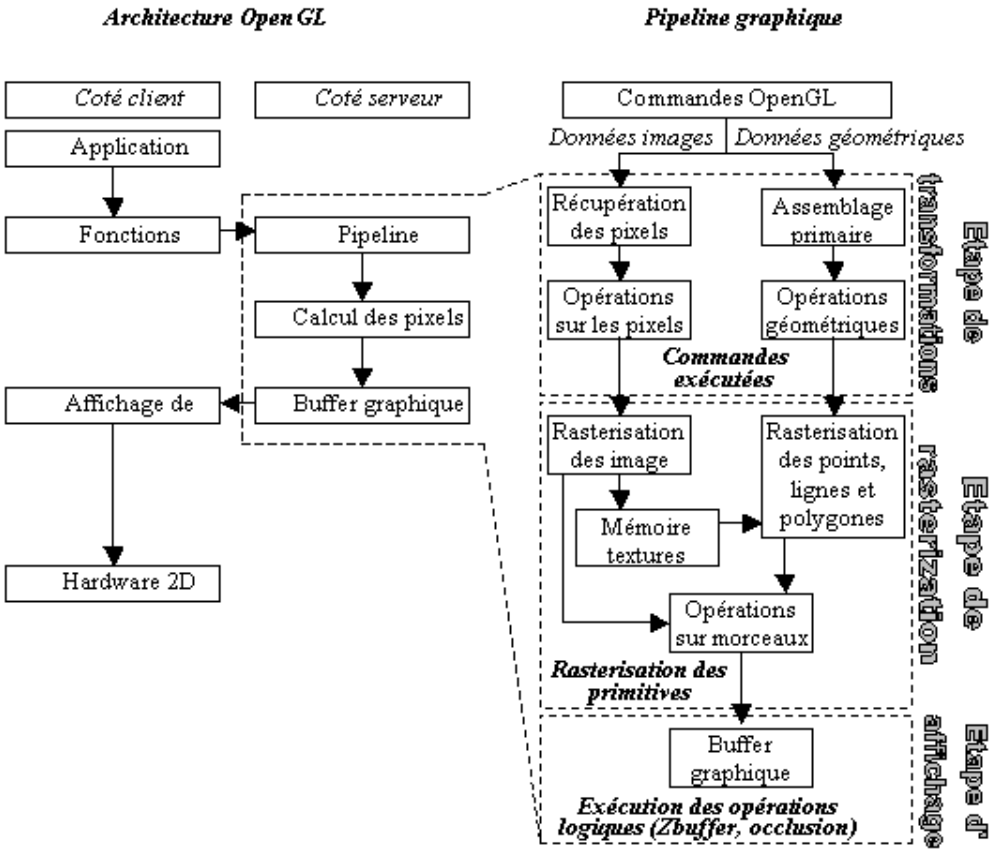


**Fig. 3.21** Architecture et pipeline d'OpenGL.

OpenGL semble pour toutes ses qualités une API de choix pour la création d'une application 3D. Le seul point négatif comparé à ses concurrents est le manque d'architecture orientée objet. Mais cela peut-être réglé en utilisant la librairie Open Inventor (également de SGI) qui est gratuite depuis peu. Contrairement à l'ancienne API de SGI, IRIS-GL, OpenGL a été créée afin de donner accès à des fonctions graphiques indépendantes du matériel et donc indépendantes du système d'exploitation utilisé. C'est pourquoi cette librairie ne propose pas de fonctions permettant la gestion des fenêtres ; cette tâche étant dépendante du système d'exploitation utilisé et plus précisément de son interface graphique. Cependant, plusieurs bibliothèques d'utilitaires peuvent s'ajouter à OpenGL pour prendre en charge cet aspect (ces bibliothèques fonctionnent également avec Mesa) :

– La librairie GLUT (*OpenGL Utility Toolkit*), considérée comme la remplaçante de la librairie AUX (*Auxiliary library*), gère de façon transparente, par procédures de type *callback*, les événements et le rendu dans des fenêtres. Elle donne la possibilité de créer des programmes pouvant être compilés et exécutés sous n'importe quel système d'exploitation possédant cette bibliothèque de fonctions. D'autre part, GLUT fournit des routines permettant de créer facilement des objets 3D complexes (des cylindres, des sphères, ...) qu'il n'est pas trivial de créer avec les quelques primitives géométriques OpenGL (points, lignes et polygones).

– La librairie GLX (OpenGL eXtension) permet de lier un programme utilisant des fonctions OpenGL avec le système graphique X Window.
– La librairie GLW (OpenGL extension for Windows) est équivalente à GLX mais pour les plates-formes Win32.

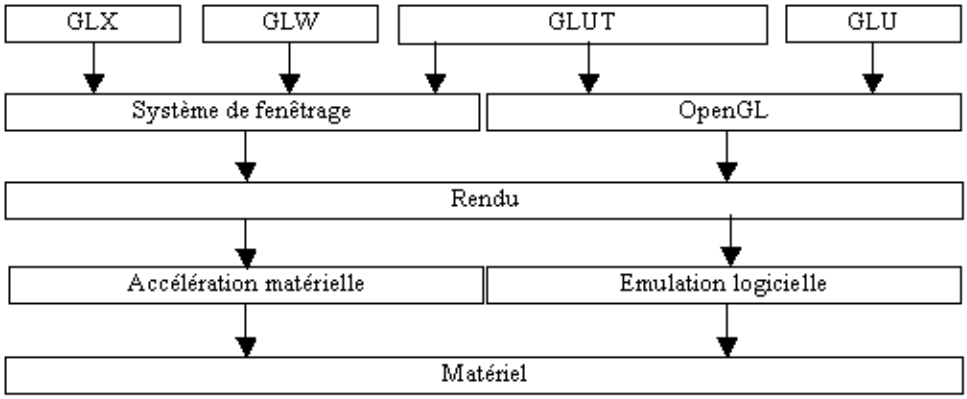


**Fig. 3.22** Position des différentes librairies OpenGL.

Il existe également la librairie GLU (*OpenGL Utility*) qui comme la librairie GLUT donne accès à des routines permettant de créer des objets 3D complexes. Cependant, GLU ne prend absolument pas en charge le système de fenêtrage. Les ensembles GLX et GLU, GLW et GLU ou AUX et GLU correspondront donc à peu près aux fonctionnalités de la librairie GLUT seule. GLU est néanmoins plus complète que GLUT pour la création d'objets complexes puisqu'elle intègre, par exemple, la tesselation.

OpenGL compte donc un certain nombre d'avantages et se place comme une librairie de choix pour créer des applications 3D :
– existe sur de nombreuses plates-formes,
– accélération matérielle complète sur les cartes graphiques haut de gamme,
– modèle client-serveur,
– existence de bibliothèques telles que GLUT ou GLU permettant de programmer une seule application pour toutes les plates-formes,
– intégration dans X Window par l'intermédiaire de l'architecture DRI (*Direct Rendering Architecture*).

Toutefois, comme nous le notons dans les désavantages donnés ci-dessous, elle s'adresse à des programmeurs habitués aux techniques 3D :
– pas de fonctions de haut niveau sans utilisation de librairies supplémentaires,
– pas de programmation orientée objet (sans l'utilisation d'OpenInventor),
– accélération matérielle douteuse sur les cartes d'entrée de gamme à but ludique.

### *Direct3D*

Direct3D est l'API 3D de Microsoft. C'est un composant de la famille d'API multimédia pour les plates-formes Win32, DirectX actuellement à la version 8.0. Pour Windows 95 et 98, ces bibliothèques dynamiques (DLL) font parties des fichiers installés alors que pour Windows NT, la version 3.0 est disponible et elle fait partie du Service Pack 5 de Microsoft. La version 7.0 a fait son apparition dans Windows 2000 et la version 8.0 est disponible en version bêta. Cette librairie 3D a été conçue principalement pour les développeurs de jeux. En effet, elle est censée remplacer la librairie WinG (*Windows Games*) qui fournissait uniquement les DIB (*Device Independant Bitmap*) capables d'être affichés rapidement (et donc utile pour le *double buffering* et les animations), chose que la GDI Windows n'est pas capable de faire. En aucun cas, WinG ne fournissait de fonctions 3D ; le pipeline 3D entier devait alors être pris en charge par le programmeur, rendant le développement d'une application 3D très coûteuse et ne tirant pas partie des possibilités du matériel sans programmation explicite.

L'important développement de cette librairie 3D est lié à l'adhésion massive des constructeurs de cartes graphiques 3D qui ont rapidement livré leur matériel avec des drivers Direct3D. Ainsi, non seulement Direct3D donne accès à des fonctions 3D gérant les différentes étapes du pipeline 3D mais, en plus, ces fonctions sont accélérées par le matériel si celui-ci en est capable. Cette propriété, bien qu'existante également pour les librairies OpenGL et Quickdraw3D[11], est plus importante avec DirectX, vu le nombre de matériels supportés.

La prise en charge du matériel sous Direct3D, DirectSound, DirectPlay, et les autres API de DirectX se fait par une couche d'abstraction matérielle HAL (*Hardware Abstraction Layer*). Cette couche permet au programmeur d'obtenir des informations et caractéristiques sur le matériel présent et également sur les performances de celui-ci. Parce que certains systèmes n'ont pas de matériel accélérant certaines fonctions 3D, une couche d'émulation HEL (*Hardware Emulation Layer*) prend en charge ces opérations particulières. Ainsi, toutes les applications peuvent bénéficier des apports de Direct3D, même si les performances s'en ressentent. L'architecture d'une application utilisant Direct3D ainsi que le pipeline graphique sont schématisés dans la figure 3.23.

Comme les autres API, Direct3D propose un mode immédiat et un mode retenu pour gérer les opérations. Le mode retenu offre une interface de haut niveau orientée objet pour la manipulation des éléments graphiques. Dans ce mode, des fonctions permettent, en particulier, de charger et de sauvegarder des fichiers décrivant les objets, les textures ou les animations.

Le mode immédiat de Direct3D peut se concevoir de deux manières différentes. Le premier est d'envoyer les commandes qui sont traitées une à une. Le second est plus considéré comme un mode retenu bis puisqu'il ressemble à s'y méprendre au mode retenu d'OpenGL. En effet, il utilise des listes de commandes comme celles d'OpenGL. Même si le nom de ces listes diffère (*display list* pour OpenGL contre *execution buffer* pour Direct3D) leur fonctionnement est néanmoins le même. A l'initialisation, les différentes commandes (les polygones et la façon dont ils doivent être traités) sont placées dans un *buffer* et lors du

[11] Ancienne API 3D pour les Macintosh avant son remplacement par OpenGL.

rendu le programme référence cette liste, au lieu de devoir spécifier toutes les commandes contenues dans cette liste à chaque affichage. La différence majeure entre les deux types de liste est leur taille. Celle d'OpenGL utilise bien moins de mémoire et se transmet facilement d'un client à un serveur alors que celle de Direct3D à tendance a saturer la mémoire vidéo.

Comme dans les modes immédiats de chacune des API présentées, celui de Direct3D ne fournit aucune fonction permettant de créer des objets complexes et de charger ou sauvegarder un fichier. Ces opérations dans ce mode sont à la charge de l'application et plus précisément du programmeur. Ce mode est particulièrement adapté aux programmeurs ayant leur propre moteur 3D mais qui souhaitent accéder à une accélération matérielle.

Le pipeline graphique de Direct3D attend en entrée un *buffer* d'exécution comme ceux mis en place dans le mode immédiat. Lorsque le programmeur travaille en mode retenu, la hiérarchie des objets constituant la scène est alors traduite en une liste d'exécutions en vue d'être passée au pipeline. Cette liste de commandes passe alors par différents modules effectuant ensemble le rendu 3D complet. Chacun de ces trois modules que nous allons présenter, est indépendant des autres. Ils sont appelés indépendamment depuis l'application Direct3D. Ainsi, le programmeur peut choisir d'utiliser ou non un de ces modules, lui préférant peut être un module spécial qu'il aura écrit pour gérer certaines options ou optimisations. Cependant, les modules standards sont particulièrement bien optimisés et donc la plupart du temps ce sont eux qui seront utilisés.

Les différents modules de Direct3D et les opérations qu'ils accomplissent sont les suivants :
- Le module de transformation est le point de départ du pipeline. Il transforme la caméra et les objets dans la liste de polygones du *buffer* d'exécution. Ce module détermine aussi quels sommets sont dans la vue (*clipping*), éliminant ceux d'entre eux qui n'y sont pas (ceux qui sont invisibles) et qui ne doivent donc pas subir le rendu final. Cette étape revêt, par ce dernier aspect, une importance capitale dans l'obtention des meilleures performances possibles pour l'ensemble du pipeline.
- Le module gérant les lumières garde une trace de toutes les lumières présentes dans la scène et détermine en fonction d'elles, la couleur de chacun des sommets des polygones. Il existe en fait deux modules différents pour gérer les lumières :
  - Le module RGB où les lumières peuvent être colorées puisqu'elles sont spécifiées par les trois canons rouge - vert - bleu. Ce mode offre bien sûr un rendu réaliste mais est également notablement moins performant si les lumières RGB ne sont pas prises en charge par le matériel.
  - Le module Ramp permet d'utiliser des lumières monochromes. Ce mode est évidemment moins réaliste mais plus économique en terme de puissance. Cependant, pour une scène comportant uniquement des lumières blanches ou grises, il convient parfaitement.
- Le module de rendu, également appelé le module de *rasterization*, affiche la scène, à partir des sommets calculés par le module de transformation et des lumières du module lumière. Il existe un grand nombre d'options pour obtenir le rendu souhaité.

En résumé, les avantages de Direct3D sont les suivants :
- accélération matérielle,
- création facile d'effets spéciaux,
- fortement modulable (possibilités de remplacer les modules d'éclairage, de rendu),

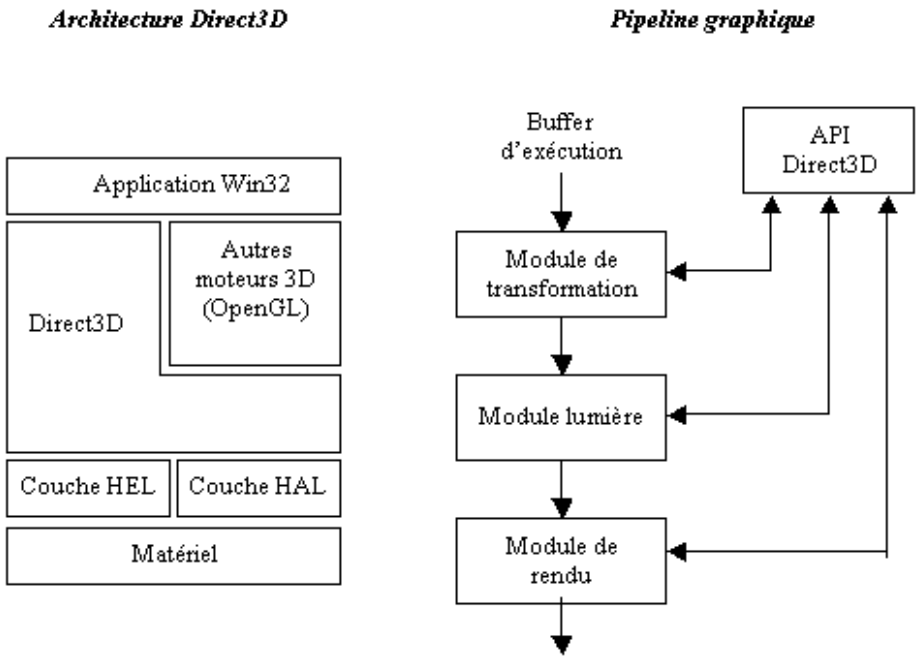


**Fig. 3.23** Architecture logicielle et pipeline graphique de Direct3D.

– programmation objet.
Mais :
– utilisation exclusive sous Windows,
– complexité de son mode immédiat.

### *Java3D*

Java3D est une nouvelle API développée par Sun Microsystems apparue au début de l'année 1998 et faisant partie de la suite JavaMedia. Cette bibliothèque d'objets donne aux programmeurs des constructeurs de haut niveau permettant de créer et manipuler une scène 3D et de construire les structures permettant de spécifier comment doit être affichée cette scène. Le principal atout de cette API est de transmettre la caractéristique de tout code en Java dite "*write once, run anywhere*", mais aussi de fournir un accès de haut niveau à des fonctions 3D performantes. Java3D est également fort bien intégrée à Internet puisque les applications et applets écrites avec Java3D ont accès à toutes les autres classes Java.

Les bases de Java3D viennent d'API graphiques existantes et de nouvelles technologies. Pour la partie bas-niveau, elle synthétise les meilleures idées des API comme Direc3D, OpenGL, QuickDraw3D tout en s'appuyant sur les deux premières. De même, pour la partie haut-niveau, elle synthétise les meilleurs aspects trouvés dans les systèmes de gestion des graphes pour les scènes. Enfin, Java3D introduit des aspects qui ne sont typiquement

pas considérés comme faisant partie des environnements graphiques, comme le son spatial 3D, mais qui immergent l'utilisateur encore plus dans la scène.

Les buts fixés pour Java3D lors de la spécification de l'API furent par ordre de priorité :
- de hautes performances : les décisions entre deux solutions possibles pour une même opération penchent toujours vers la solution la moins coûteuse, favorisant ainsi le temps d'exécution.
- un riche choix de fonctions : la difficulté fut de trancher entre les fonctions permettant de créer des scènes intéressantes et celles qui paraissaient non essentielles voire obscures.
- programmation orientée objet : offrir un paradigme de POO de haut niveau qui permette aux programmeurs de développer rapidement des applications et des applets sophistiquées. Une application consistera à créer différents objets graphiques et à les connecter entre eux pour former une structure arborescente appelée graphe de scène.
- chargement des fichiers 3D : fournir des fonctions de chargement de divers formats de fichier de description de scène 3D, autorisant ainsi l'utilisation d'un grand nombre de fichiers du monde de la CAO et entre autres VRML.

Java3D propose plusieurs mécanismes pour le rendu des scènes : le mode immédiat, le mode retenu et le mode retenu compilé. Chacun de ces modes offre, par ordre, de plus en plus de liberté au moteur Java3D pour les optimisations internes lors de l'exécution de l'application :
- Comme pour les API précédentes, le mode immédiat sert à donner des primitives de base que le moteur 3D doit traiter immédiatement. Même si la place réservée normalement aux optimisations dans ce mode de rendu est petite, Java3D accélère cependant le rendu de chacun des objets, grâce à un rehaussement du niveau d'abstraction. En effet, il est permis dans ce mode, d'utiliser ou non un graphe de scène et d'utiliser des fonctions des deux modes retenus. Pour les deux modes suivants, l'utilisation du graphe de scène est obligatoire puisque c'est cette structure qui renferme (qui retient) les objets graphiques à dessiner.
- Le mode retenu est pratiquement aussi flexible que le mode immédiat en augmentant néanmoins sensiblement les temps de rendu. Le graphe de scène est entièrement manipulable. L'ajout, l'effacement et l'édition d'un nœud sont des opérations permises dont la mise à jour est immédiate à l'écran.
- Le mode retenu compilé autorise le moteur Java3D à lancer une série d'optimisations complexes comme la compression des objets graphiques, l'aplanissement du graphe de scène, le groupement des objets, ... Une fois compilées, le programmeur a un accès minimal aux structures des objets et du graphe lui-même. Il peut accéder à certaines propriétés uniquement s'il a spécifié les propriétés qu'il veut pouvoir modifier grâce à des variables booléennes.

Le modèle de programmation orientée objet de Java3D, basé sur les graphes de scène, fournit un mécanisme simple et flexible pour représenter et pour afficher tous les éléments d'une scène. Cette API autorise en effet l'utilisateur à penser en terme d'objets plutôt qu'en ensembles de triangles et également à la composition de la scène plutôt qu'à la façon d'en effectuer le rendu. C'est en cela que Java3D est une API de niveau supérieur. Le graphe de scène est constitué d'objets Java 3D, appelés nœuds, arrangés dans une structure arborescente. L'utilisateur de l'API Java 3D crée un ou plusieurs sous-graphes de scène et les attache à un univers virtuel (objet VirtualUniverse). La seule restriction de Java 3D

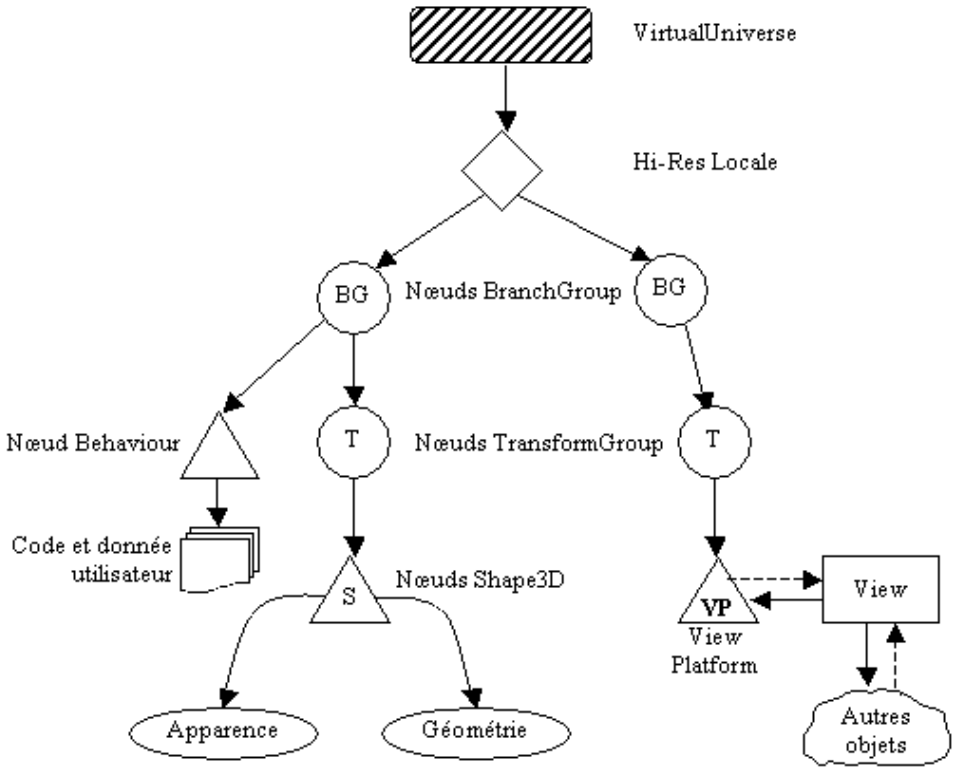


**Fig. 3.24** Exemple simple de graphe Java3D.

est de ne pas supporter des cycles dans le graphe. Les graphes et sous-graphes d'une scène Java 3D seront donc des graphes directs acycliques tout comme dans une hiérarchie de nœuds VRML.

Dans Java 3D, les nœuds sont séparés en deux catégories, les groupes et les feuilles, comparables aux classes *container* et *component* de Java. Les groupes contiennent un ou plusieurs nœuds fils mais ne peuvent avoir qu'un seul nœud parent. Les feuilles, quant à elles, contiennent les définitions des objets (géométries, lumières, sons, etc.) présents dans la scène. Ces nœuds feuilles ne peuvent, en aucun cas, contenir des nœuds fils et n'ont également qu'un seul parent.

L'organisation hiérarchique du graphe de scène autorise naturellement le groupement spatial des objets graphiques se trouvant aux feuilles du graphe. Tous les nœuds internes jouent ainsi un rôle de regroupement de leurs nœuds enfants mais également celui de frontière spatiale englobant tous les objets géométriques définis par ses descendants. Cette dernière propriété est d'une grande utilité pour l'implémentation efficace de certaines opérations comme la détection de proximité, la détection de collision, ...

L'état d'une feuille est défini par tous les nœuds se trouvant sur le chemin direct menant de cette feuille au sommet du graphe de scène. Parce que le contexte graphique d'une feuille ne

dépend que de ce chemin, le moteur Java3D peut décider de parcourir le graphe de scène dans un ordre quelconque (profondeur d'abord, largeur d'abord ou même en parallèle). Les seules exceptions sont les propriétés bornées dans l'espace comme les lumières ou le brouillard. Cette propriété est cependant un changement par rapport à des API plus vieilles comme PHIGS ou SGI Inventor où le changement d'état d'un nœud affectait tous les nœuds en dessous et à droite de ce nœud dans l'arbre.

Java3D tire la plupart de ses qualités du langage Java mais aussi de part sa conception :
- multi-plateformes (*"write once, run anywhere"*),
- utilisation de graphes de scène,
- s'appuie sur OpenGL et Direct3D pour la couche bas-niveau et profite donc de l'accélération offerte par les cartes graphiques 3D,
- intégration facile, grâce aux nombreuses bibliothèques Java existantes, d'éléments multimédias, de gestion réseau, de reconnaissance vocale, ...
- son 3D.

Mais, compte à son détriment :
- des performances laissant à désirer,
- n'existe que pour Window et IRIX et plus récemment sous Linux.

### *Farhenheit*

SGI et Microsoft ont formé un groupe de travail pour définir une architecture graphique qui sera sans aucun doute l'avenir des API graphiques 3D. Ce projet du nom de Fahrenheit a pour but de fournir aux développeurs un outil puissant permettant de créer facilement des applications 3D sur les plates-formes SGI, Windows et HP-UX.

L'API Fahrenheit comprend 3 composants :
- L'API FLL (*Fahrenheit Low Level*) est la librairie de rendu bas niveau supportant l'abstraction du matériel graphique. Cette API travaille avec les primitives graphiques de base (lignes, points, triangles). Les applications qui utilisent cette API devront, comme avec OpenGL et le mode immédiat de Direct3D, spécifier au système comment dessiner une scène avec les primitives graphiques et leurs propriétés. FLL remplacera l'API Direct3D sous Windows, se plaçant à côté d'OpenGL qui sera conservée.
- L'API FSG (*Fahrenheit Scene Graph*) est l'API qui fournit aux programmeurs un plus grand niveau d'abstraction en s'appuyant sur la couche inférieure (OpenGL, Direct3D ou FLL). Elle permet de décrire une scène sous forme de graphe composé d'objets comme les formes géométriques, les textures, les lumières, etc. Les applications utilisant cette API devront décrire ce qu'il faut dessiner plutôt que comment dessiner. Il sera également possible (et c'est ce qui est nouveau par rapport aux graphes Java3D) de spécifier quelles sont les préférences de l'application : la qualité d'image ou un taux de rafraîchissement fixe. Cette dernière possibilité offre de nouvelles perspectives dans l'utilisation de la 3D pour l'affichage de processus en temps réel. Le graphe de scène géré par FSG est optimisé et les performances devraient être étonnantes.
- L'API FLM (*Fahrenheit Large Model*) sera une extension de FSG permettant de créer et de visualiser de larges modèles d'informations à partir de constructions typiques de la CAO comme les surfaces paramétrées NURBS (*Non Uniform Rational B-Splines*).

La figure 3.25 montre l'architecture de Fahrenheit et comment les différentes API peuvent être utilisées dans une même application. Cela permet, d'une part, d'utiliser une spécification haut-niveau pour une scène tout en ajoutant des appels de fonctions bas niveau pour des raisons spécifiques. Pour les applications actuelles, cela permet, d'autre part, d'adopter petit à petit les API FSG et FLM plutôt que de devoir réécrire les applications en entier.

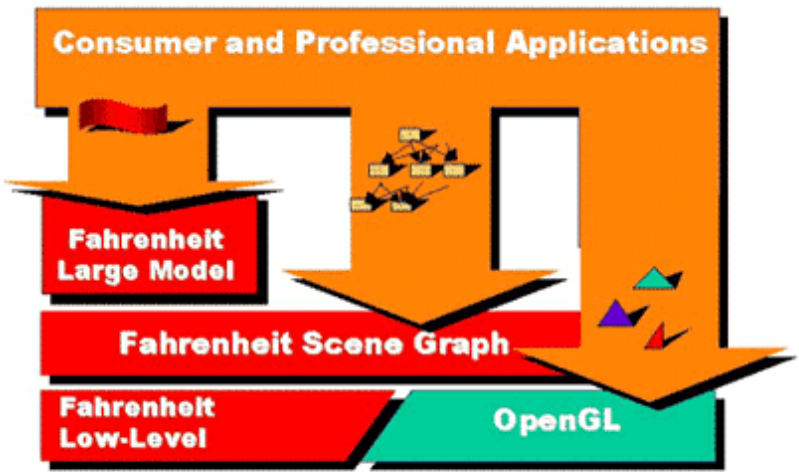


**Fig. 3.25** L'architecture de Fahrenheit.

Le composant le plus important dans l'architecture Fahrenheit est l'API FSG. Elle permet de supporter un grand nombre de configurations matérielles et de répondre aux préférences d'une application. Ses points forts sont les suivants :

- Une alternative face à la philosophie générale de la 3D dite AFAP (*As Fast As Possible*)
- L'abstraction des primitives graphiques tout en maintenant de très bonnes performances. Les programmeurs d'interfaces 3D pourront se concentrer sur les fonctionnalités de leurs applications plutôt que de résoudre les problèmes liés à la programmation en utilisant une API de bas niveau.
- L'ajout ou la modification des données du graphe de scène en temps réel.
- Le support des machines multiprocesseurs sans devoir inclure un code spécifique. Différents processus peuvent au graphe de scène simultanément et le modifier. Les tâches courantes comme le rendu, les calculs d'intersections et de collisions bénéficient automatiquement de la présence de plusieurs processeurs.

Pour l'instant, Fahrenheit n'est pas encore une librairie 3D comparable à OpenGL ou Direct3D. En effet, c'est plutôt une librairie d'émulation puisqu'elle s'appuie sur les deux API citées précédemment pour l'accélération matérielle. Or, tout comme Java3D, nous nous méfions des API d'émulation du point de vue performance.

## Les moteurs 3D pour Internet

Les fonctions d'un moteur 3D pour Internet (quel qu'il soit) sont de charger une scène décrite à l'aide d'un langage support, de la dessiner à l'écran et de s'occuper des animations et des interactions de l'utilisateur. Pour effectuer ces différentes tâches, un navigateur comporte plusieurs sous-systèmes comme ceux donnés dans la figure 3.26 (ces sous-systèmes sont seulement représentatifs et peuvent différer légèrement d'un navigateur à l'autre).

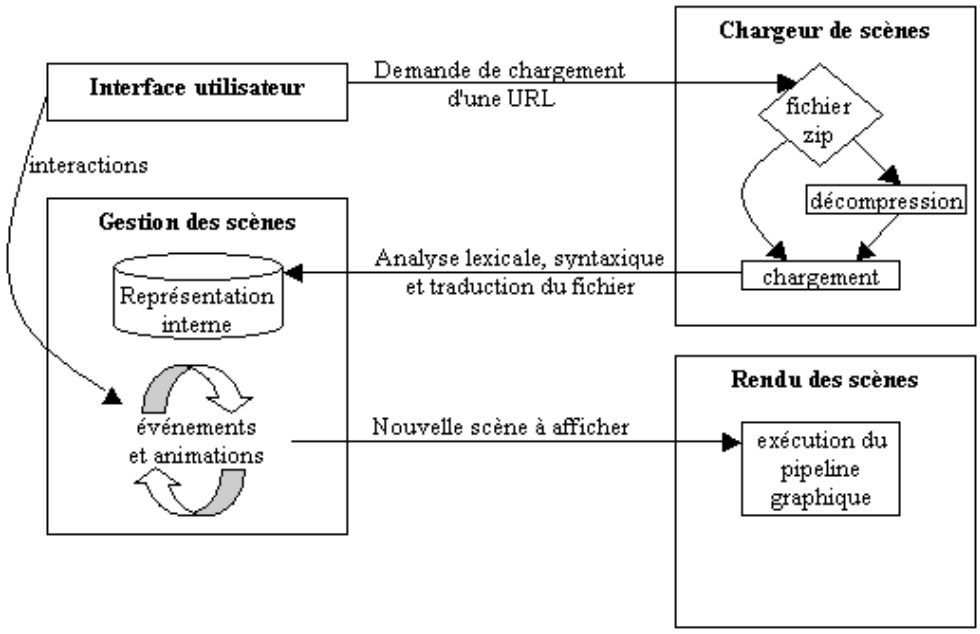


**Fig. 3.26** Les composants d'un co-navigateur 3D.

Le sous-système "interface utilisateur" gère le comportement de l'emplacement occupé par la scène dans une page Web. Ce sous-système positionne également les boutons, les menus et l'espace qu'occupera la scène 3D. Il répond également aux événements de l'utilisateur (frappe au clavier, sélection d'un élément d'un menu, déplacement ou clique de la souris, etc.) ; c'est-à-dire que lorsque l'utilisateur interagit avec la scène, c'est ce sous-système qui répond aux événements et qui en avertit les autres sous-systèmes.

Le sous-système "chargeur de scènes" gère, comme son nom l'indique, tout ce qui concerne le chargement d'une scène. Il gère plus précisément le chargement de la scène elle-même mais également celui des textures, des sons, des prototypes externes ou des scripts qui y sont utilisés. C'est ce système qui gère les URL et émet les erreurs lorsque quelque chose ne va pas. Il fournit au sous-système de "gestion des scènes" une représentation interne de la scène.

Le sous-système "gestion des scènes" gère la représentation des scènes. Les structures internes au navigateur incluent notamment des informations qui permettent de dessiner les scènes plus vite. Ce sous-système garde également une trace du point de vue de l'utilisateur, des caractéristiques de navigation et gère les événements lorsque l'utilisateur entre en collision avec un objet solide. De plus, le sous-système "gestion des scènes" garde un état des différents capteurs, scripts, sons et animations qui sont déclarés. Avec tout ce travail à effectuer, ce sous-système est le centre nerveux d'un navigateur 3D. Les calculs à effectuer par ce sous-système sont effectués lorsque celui de rendu des scènes ne travaille pas (*idle* – inoccupé en français).

Le sous-système "rendu de scènes" fonctionne avec le sous-système gestion des scènes pour convertir la représentation interne de la scène en une image à l'écran. Pour ce faire, ce sous-système récupère périodiquement la description de la scène et la redessine à chaque fois que l'utilisateur bouge le point de vue ou qu'une animation de la scène modifie un élément. Ce sous-système contient tout le pipeline graphique et effectue par conséquent la grande majorité des calculs. C'est donc le principal coupable lorsqu'on remarque qu'une scène n'est pas rafraîchie assez vite.

Toutes les solutions techniques présentées ici fonctionnent sur le même modèle. Elles permettent donc toutes de mettre facilement en ligne (sur Internet ou Intranet) une scène 3D. Elles sont peu lourdes en programmation puisqu'elles utilisent des moteurs existants (nécessitant uniquement l'écriture d'un script CGI pour générer la scène si elle doit être produite dynamiquement). Cependant, tous ces moteurs de rendu sont à usage général et par conséquent non optimisés pour une application particulière. En général, l'ajout de fonctionnalités non prévues est également problématique car nécessitant une programmation lourde.

### *VRML*

VRML (*Virtual Reality Modeling Language*) est un format de fichier qui, associé à un *plugin* (application augmentant les possibilités d'un navigateur HTML), forme un système de consultation d'un contenu 3D. Pour éviter un amalgame souvent fait par abus de langage, le véritable moteur 3D dans ce système est le *plugin* et non le "langage" VRML. En effet, VRML n'est pas un langage de programmation, même s'il peut intégrer des scripts. Ce n'est qu'un format de fichier permettant de décrire des scènes 3D interactives avec possibilité d'y inclure des liens sur d'autres contenus consultables depuis un navigateur.

VRML 2.0 est la dernière version du langage même si son nom est devenu VRML97 depuis qu'elle a été reconnue comme standard international par l'organisme de normalisation ISO en décembre 1997 sous le numéro ISO/IEC 14772-1 :1997 [VRML97]. La norme ne définit pas de dispositifs physiques à utiliser ou de concepts dépendants du système comme la résolution d'un écran. Ainsi, le format de fichier VRML reste totalement indépendant des éléments d'un système et par conséquent est multi-plateforme (dans la mesure où le *plugin* existe sur les plates-formes visées). L'évolution de VRML entre la première et la deuxième norme touche principalement les points suivants :
– sondeurs de proximité, de visibilité, ...,
– animations automatiques,

- des scripts peuvent être intégrés dans le fichier VRML,
- textures fixes ou animées,
- son 3D,
- collisions.

Ces nouvelles fonctionnalités permettent d'atteindre les qualités suivantes nécessitant, pour la plupart, uniquement de les décrire et non de les programmer :

- Un monde réaliste :
  - On ne traverse pas les murs.
  - Image de fond (ciel).
  - On suit le relief du terrain grâce à la gravité.
  - Le temps passe.
- Un monde animé :
  - Des éléments peuvent bouger dans la scène (des oiseaux volent, des voitures roulent) ! ;
  - Ils peuvent changer de couleur ou de forme.
- Un monde interactif :
  - Les objets peuvent être déplacés.
  - On peut les faire réagir en les touchant avec la souris ou en passant à proximité (un chien aboie lorsque l'on s'en approche, ouvrir une porte, déclencher un son).
- Un monde connecté :
  - On peut rencontrer un avatar et dialoguer avec lui.
  - On peut créer des liens permettant d'aller sur un autre site en cliquant sur un objet dans la scène.
  - On peut créer un monde composé d'objets venant d'autres sites (répartition des données sur différents serveurs HTTP).
- Un monde multimédia :
  - Intégration du son à une scène.
  - Intégration de textures animées.

Avec VRML, la description d'une scène est relativement aisée. De nombreux outils (modeleurs professionnels ou spécialisés) permettent de générer une scène riche. Cependant, tout ce qui concerne la gestion des interactions dans la scène est relativement difficile à intégrer dès lors que les comportements des objets sortent des possibilités de VRML. Cela ne veut pas dire que l'intégration de ces comportements est impossible mais ils nécessitent la programmation de scripts en Java ou en JavaScript. Avec ces deux langages, le graphe de scène VRML est entièrement accessible et modifiable.

A l'aide de ces scripts, il est donc possible de réaliser des opérations non incluses dans la spécification du langage VRML. Ils permettent de rendre dynamique une scène VRML normalement statique. De plus, avec le langage Java en particulier, nous pouvons accéder à de nombreuses librairies (classes) pour effectuer des opérations diverses depuis un script VRML. Il sera ainsi possible, par exemple, de récupérer l'état des capteurs (par l'implémentation d'un système de consultation en Java) et d'afficher ces informations dans le graphe de scène.

### *MPEG4*

MPEG4 est l'évolution naturelle, à la vue des possibilités multimédias actuelles, des précédentes normes MPEG (*Moving Pictures Expert Group*). Elle répond aux besoins d'intégration des différents objets multimédias et d'interactions sur ceux-ci dans un contexte multi-utilisateurs pour le travail coopératif. Pour cela, MPEG4 fournit des outils de diffusion et d'accès aux informations multimédias sur différents types de réseaux homogènes ou hétérogènes, avec ou sans pertes de données. Cette norme qui veut se placer comme un standard pour le codage d'informations multimédias, intègre également la possibilité d'interagir sur les objets diffusés. Elle permet la transmission d'images et de sons, naturels ou synthétiques, 2D et 3D que l'on nomme AVOs (*Audio Visual Objects*).

MPEG4 n'est pas, contrairement aux autres normes MPEG, qu'une définition d'un format de fichier mais une architecture logicielle complète reposant sur des spécifications détaillées pour le traitement des flux multimédia. Cette spécification inclut notamment les différentes études suivantes :
- les méthodes de codage des AVOs,
- la méthode de codage d'une scène complète composée d'AVOs agencés sous forme de graphe,
- le multiplexage et la synchronisation des différents flux de données contenant les AVOs,
- l'interface de communication entre les clients et le serveur MPEG4.

Pour gérer ces différents aspects se plaçant à des niveaux différents de la chaîne de restitution de l'information véhiculée sous le format MPEG4, l'architecture logicielle peut être présentée comme un ensemble de trois couches (que l'on peut voir schématisées dans la figure 3.27) :
- La couche de distribution assure le transport des différents flux d'informations jusqu'à l'application finale en faisant abstraction du type de réseau utilisé. Elle comprend les fonctions de multiplexage et de démultiplexage des différents flux élémentaires et la sous-couche DMIF (*Delivery Multimedia Integration Framework*). Cette sous-couche DMIF définit les interfaces de liaison avec le réseau, DNI (*DMIF Network Interface*), et avec l'application, DAI (*DMIF Application Interface*).
- La couche système donne accès aux mécanismes d'identification, de description et d'association des différents flux de données et permet la synchronisation entre les différents objets composant une scène.
- La couche compression quant à elle effectue la compression/décompression de la scène et des différents objets la composant. La scène est codée sous la forme d'un arbre de nœuds comme en VRML, auquel ont été ajoutées des fonctionnalités pour la gestion des flux audio et vidéo 2D. Ce graphe d'objets est ensuite compressé sous un format binaire appelé BIFS (*Binary Format For Scenes*). Chacun des objets de la scène est également compressé par son propre format de compression (les vidéos en MPEG2 par exemple). Les objets peuvent même utiliser une compression non reconnue par l'application puisque le décodeur peut être transmis dans le flux de données décrivant l'objet.

Cette nouvelle norme MPEG semble, à première vue, s'adapter parfaitement à la création d'applications 3D interactives. Cependant, elle est encore relativement floue. La plupart des informations et des applications sont uniquement accessibles par les membres du groupe d'experts. La spécification est néanmoins alléchante puisqu'elle permet de gérer en théorie,

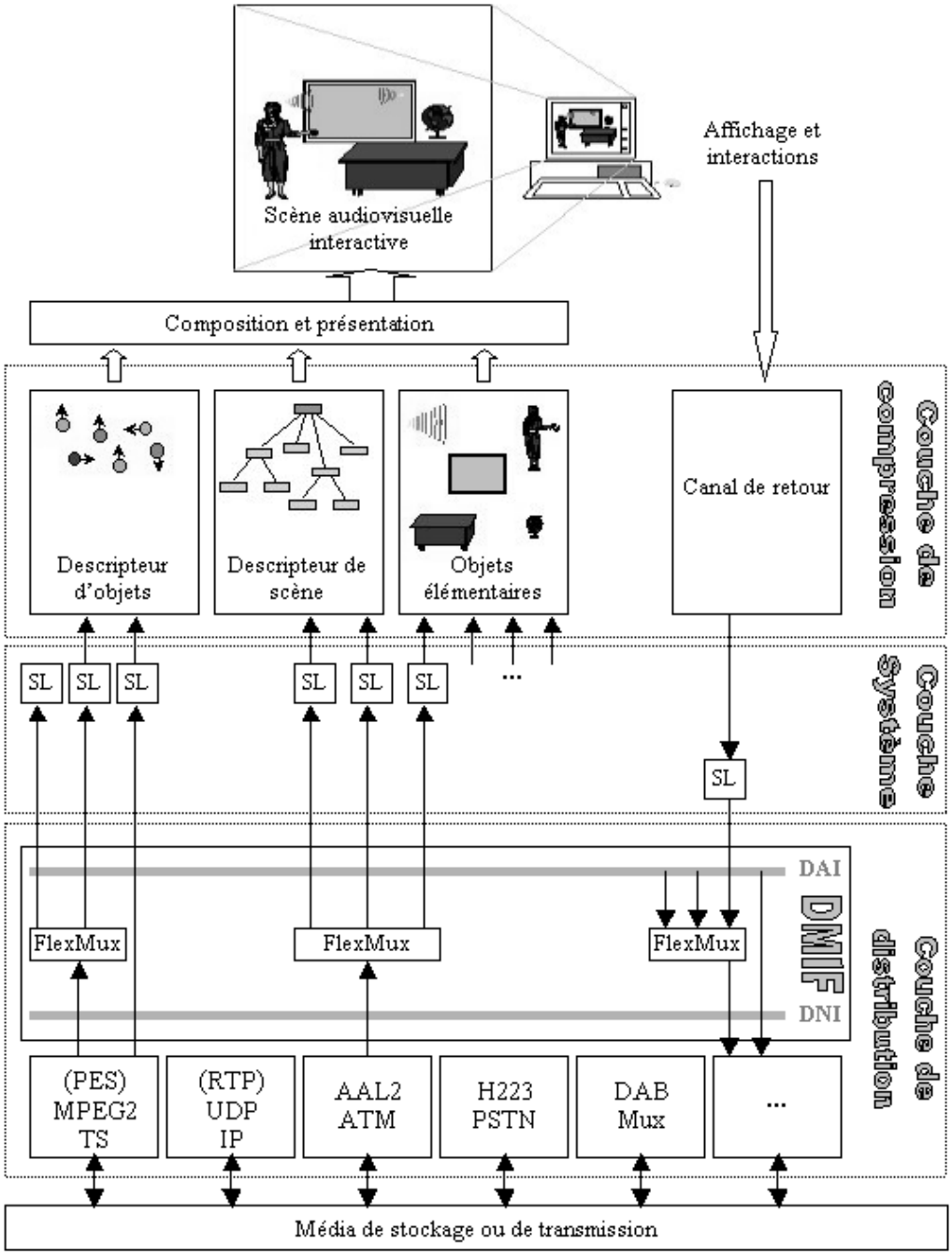


**Fig. 3.27** Les différentes couches de l'architecture MPEG4.

tous les aspects du multimédia. Mais nous avons l'impression que MPEG4 est à la diffusion de contenu, ce qu'est MPEG7 à leur description ; c'est-à-dire une très belle norme très peu utilisée, voire peu utilisable. Il nous est par conséquent impossible d'évaluer les possibi-

lités 3D de MPEG4. Cependant, d'après les informations présentes dans les documents [AVARO97] et [MARQUES98], nous pouvons d'ores et déjà prévoir ses avantages pour la 3D :
- la possibilité de réserver une qualité de service pour la transmission des flux de données,
- la compression par un facteur 10 de la scène, devrait faire gagner un temps précieux lors de la récupération d'un document,
- la forte intégration des différents composants multimédias,
- la possibilité de créer des scènes multi-utilisateurs avec un avatar pour chacun des membres.

En résumé, MPEG4 compte tous les avantages de VRML auxquels s'ajoutent :
- le partage des données,
- la synchronisation entre les différents flux de données (son, vidéo, 3D, ...).

Mais comporte les points négatifs suivants :
- la norme pas encore mure,
- la gestion de la scène est centralisée,
- ne dispose pas encore d'outils pour évaluer les ressources matérielles nécessaires.

### *Metastream*

MetaCreation propose son format de fichier pour décrire des données 3D. Les avantages principaux de ce format de fichier et de son *plugin* associé sont les suivants :
- La taille moyenne des fichiers Metastream est plus petite que n'importe quel autre format de fichiers 3D. Il en résulte un transport plus rapide des données sur le Web.
- Le *plugin* ajuste automatiquement les attributs des objets afin d'obtenir un rapport qualité/vitesse optimal.
- L'affichage des objets commence dés le début du chargement des premières faces (d'où *streaming* : le contenu s'affine avec l'arrivée de nouvelles informations dans le flux de données).

L'inclusion dans une page HTML d'une URL vers un fichier Metastream se fait par la balise `<OBJECT>` ou la balise `<EMBED>`. Ce format de fichier représente un travail intéressant puisqu'il apporte une solution au problème des fichiers non compressés comme VRML. D'autre part, il offre, pour la 3D, une solution à un problème fondamental d'Internet : la non-connaissance du matériel du client. Les créateurs de sites doivent, en effet, toujours considérer le public qu'ils visent en fonction des technologies utilisées. Dans le cas de la 3D, mais cela peut être généralisable à tous les médias gourmands en ressources, l'impossibilité de connaître les possibilités graphiques d'un client rend la création difficile puisqu'il faut trouver un compromis entre la qualité de la scène et son temps de rendu chez les clients. La première solution consiste à réduire la qualité d'une scène (moins de polygones, moins de textures, moins d'animations, ...) afin de créer un contenu que la majorité des clients pourront lire. La deuxième est de créer un contenu pour une certaine classe de client, ceux ayant les plus grandes capacités. Dans les deux cas, le créateur est frustré puisque la scène qu'il diffuse n'est pas de la qualité souhaitée, ou elle ne s'adresse qu'à un petit nombre d'utilisateurs. Metastream apporte un remède à ce problème puisqu'une scène peut être déposée sur Internet avec une qualité optimale. Ceci, en fonction de paramètres matériels

tels que le processeur de la machine, les possibilités graphiques ou la bande passante de la liaison Internet, il ajuste automatiquement la taille de la scène et les options de rendu.

La dernière version du *plugin* Metastream, MTS3, offre les mêmes avantages que les formats précédents : prise en compte du matériel du client pour le choix des options de rendu et *streaming* du flux de données 3D. Un effort particulier a été fait pour améliorer le réalisme des scènes. Ainsi les options suivantes ont été ajoutées par rapport à d'autres *plugins* :
- Antialiasing de la scène en temps-réel : l'image en résultant (la trame courante) est adoucie par l'utilisation de filtres.
- Antialiasing progressif : l'image est raffinée en plusieurs passes (de 2 à 32) pour obtenir une image de qualité photographique.
- Projection des ombres en temps-réel : l'ajout d'ombre fournit une information supplémentaire sur la position d'un objet et permet d'appréhender d'une façon plus efficace sa taille.
- Coloriage Phong en temps-réel : la réflexion des lumières de ce modèle de coloriage rend les scènes plus réalistes.
- Intégration dans une page HTML : le fond de la scène peut être transparent pour que le contenu d'une page HTML se trouvant en arrière plan soit toujours visible.

Les scènes sont "scriptables" et un kit de développement est également disponible pour inclure des scènes MTS3 dans une application. Metastream s'avère être donc plus complet que VRML tant pour la qualité de rendu que pour les possibilités d'intégration.

### *Scol et Blaxxun*

Scol est un langage de communication, une sur-couche de TCP/IP, développé par la société Cryo pour le projet "le deuxième monde" commun avec Canal+, qui établit une connexion directe et permanente entre deux points. Non seulement des données sont échangées par ce protocole mais il permet également d'exécuter des fonctions à distance, dés lors que les deux machines l'autorisent. C'est là qu'entre en jeu le moteur 3D puisque parmi ses fonctions intégrées à Scol apparaissent des routines 3D.

En regardant de plus près les différentes possibilités de Scol, on s'aperçoit que ce langage est un véritable langage de programmation multimédia intégrant des fonctions permettant de gérer des aspects variés :
- l'environnement, les fichiers,
- les graphiques 2D et 3D,
- le son et la vidéo,
- la liaison avec les navigateurs HTML,
- les communications réseaux, les protocoles SMTP[12] et HTML,
- les bases de données.

Scol est disponible en *plugin* pour les navigateurs Netscape et Internet Explorer versions 3.0 et supérieures. L'avantage de ce moteur par rapport à Viscape (un autre moteur 3D pour le web), outre qu'il intègre de nombreuses fonctions, est que de multiples outils et un kit pour les développeurs sont disponibles, gratuitement, avec leur documentation.

[12]Simple Mail Transfert Protocol.

Aux dires de Philippe Ulrich, fondateur de Cryo, lors d'une conférence de présentation du "deuxième monde" au Conservatoire National des Arts et Métiers dans le cadre des séminaires du D.E.A. médias et multimédia, le langage Scol était destiné à être le moteur du microcosme virtuel. Cependant, en allant sur le site officiel du "deuxième monde", nous nous rendons compte que le moteur actuel est Blaxxun et non Scol. Et d'ailleurs, sur le site de Cryo, aucune phrase, aucun lien ne mentionne l'existence de ce projet commun avec Canal+, mais plutôt, une communauté virtuelle appelée Cryopolis.

Blaxxun propose une série de produits tous dans le domaine de la communication multimédia sur Internet. L'ensemble de ces produits offrent a peu près les mêmes possibilités que le langage Scol :
- Blaxxun Contact 4.2 est le programme client gratuit permettant de visiter les sites hébergeant des environnements virtuels. Comme Scol, ce client intègre de nombreuses fonctions multimédias dont la possibilité d'afficher des scènes 3D au format VRML ou Viscape. Ce client existe sur les plates-formes Win32 sous la forme d'un *plugin*.
- Blaxxun Community Server 4.0 est le programme serveur payant mais disponible en version d'évaluation sur le site de Blaxxun. Il permet de mettre en ligne des applications multimédias mêlant les aspects de communauté, de commerce et de collaboration.
- Blaxxun Community Platform SDK 4.0 est le kit de développement servant à créer une application "sur mesure" utilisant les produits Blaxxun. Il permet de modifier le côté serveur et le côté client.

### *Virtools Dev*

Les langages tels que Scol ou Blaxxun bien que très complets, au niveau de leurs fonctionnalités, ne s'adressent qu'à des programmeurs ayant des connaissances de la 3D. Pour parvenir à toucher le plus grand nombre de concepteurs, une nouvelle génération de langages est apparue il y a deux-trois ans. Ces langages se séparent en deux catégories :
- Les langages scriptés tels que Alice [Conway94] utilisent une syntaxe proche du langage naturel. On y manipule des notions de plus haut niveau du type "Joe suit Jack" plutôt que de manipuler les opérateurs mathématiques de transformations classiques comme les matrices.
- Les langages visuels permettent une construction interactive des scènes 3D en reliant des boîtes comportementales. Tout comme Max et JMax proposent une construction visuelle des traitements sonores [Dechelle98], ces outils visuels 3D permettent de mettre en place une chaîne de traitements 3D.

Les premiers, bien que disposant de fonctions simplifiées, s'appuient sur des langages de programmation nécessitant une connaissance informatique minimale. On accède aux fonctionnalités d'Alice, par exemple, à travers le langage Python, lui-même employant une syntaxe proche de la syntaxe C++. Ces langages sont, par conséquent, plus orientés vers les programmeurs novices en 3D. Les langages visuels, comme Cult3D Designer et Virtools Dev, élargissent encore plus ce principe de simplification de la conception des scènes 3D. Ils mettent en place pour cela une interface permettant de construire et manipuler un graphe de traitements par boîtes comportementales. Par la liaison visuelle de ces différents comportements, ils remplacent totalement la création de codes sources. Bien que simplifiant et rendant la création de scènes 3D accessible à un plus grand nombre de concepteurs, ils

nécessitent cependant des connaissances minimales des techniques 3D et un certain apprentissage. Nous nous intéressons par la suite au logiciel Virtools Dev qui nous a servi à prototyper notre dernière génération d'interface d'accès aux bibliothèques numériques (voir section 4.4.6 page 163).

Virtools Dev est un atelier de création d'applications interactives mêlant la 3D temps réel, le son et de la vidéo numérique. La figure 3.28 fournit une copie d'écran de l'interface de Virtools Dev. Ce logiciel peut être comparé aux environnements de développement standard permettant de simplifier considérablement la création d'applications. Cependant, alors que dans un IDE (*Interface Developement Environment*) le code produit automatiquement doit être complété, avec Virtools Dev, l'application est entièrement générée. Si d'aventure, un concepteur veut créer une application nécessitant un comportement sur mesure, il peut le faire grâce à l'API fournie et l'incorporer dans les scripts de son application.

La première étape est l'importation de scènes créées dans un modeleur 3D (3DS Max ou MAYA). Ces scènes peuvent contenir des animations et des textures que Virtools Dev convertit grâce à certains *plugins* spécifiques. La deuxième phase concerne la mise en place des interactions grâce aux scripts visuels. Pour ce faire, il suffit d'appliquer des comportements prédéfinis à certains objets de la scène. Les objets et leurs comportements sont ensuite affichés, animés et réactifs en temps réel à l'aide des procédures de Direct3D ou d'OpenGL.

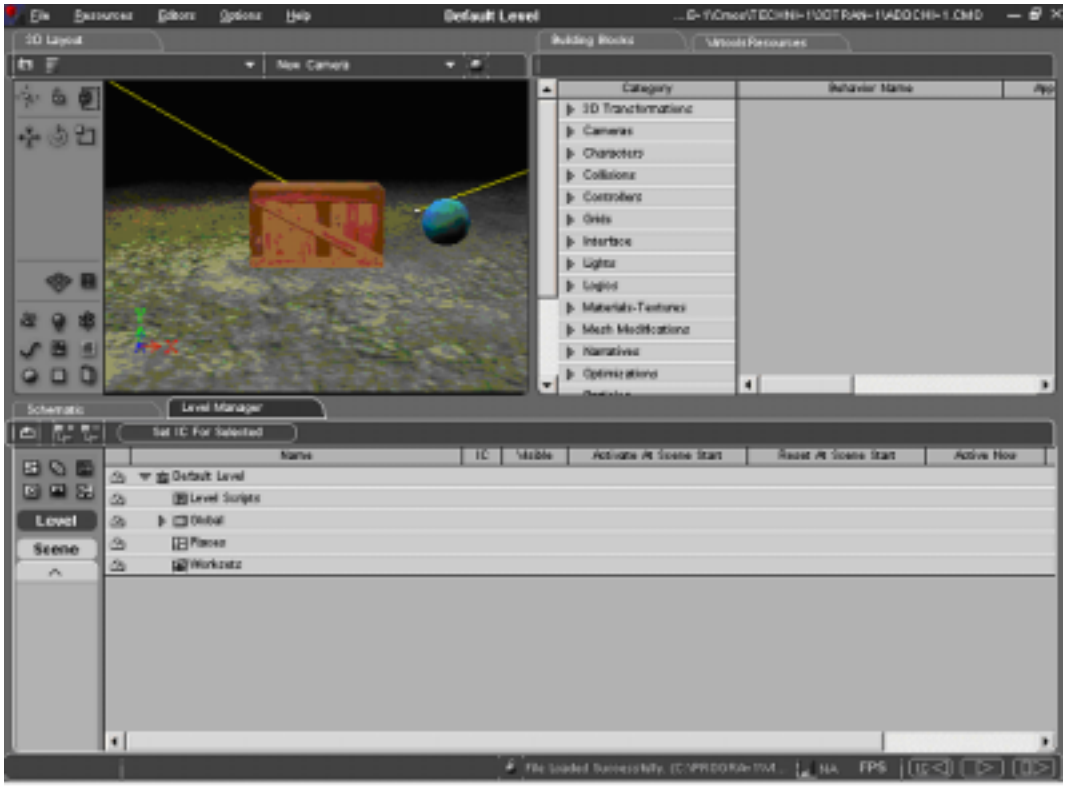


**Fig. 3.28** Une copie d'écran du logiciel Virtools Dev.

Les interactions se composent à l'aide de *Building Blocks*, ou briques comportementales, que l'on dépose par simple copier-déposer sur l'objet choisi. Cette opération crée un nouveau script pour l'objet en question. Les comportements peuvent être reliés entre eux, comme dans la figure 3.29, au sein d'un même script de façon à construire le scénario de l'objet. Ils peuvent également être reliés à d'autres scripts ou d'autres objets, déclenchant ainsi des opérations sur d'autres objets, voire d'autres scènes pour constituer le scénario global de l'application.

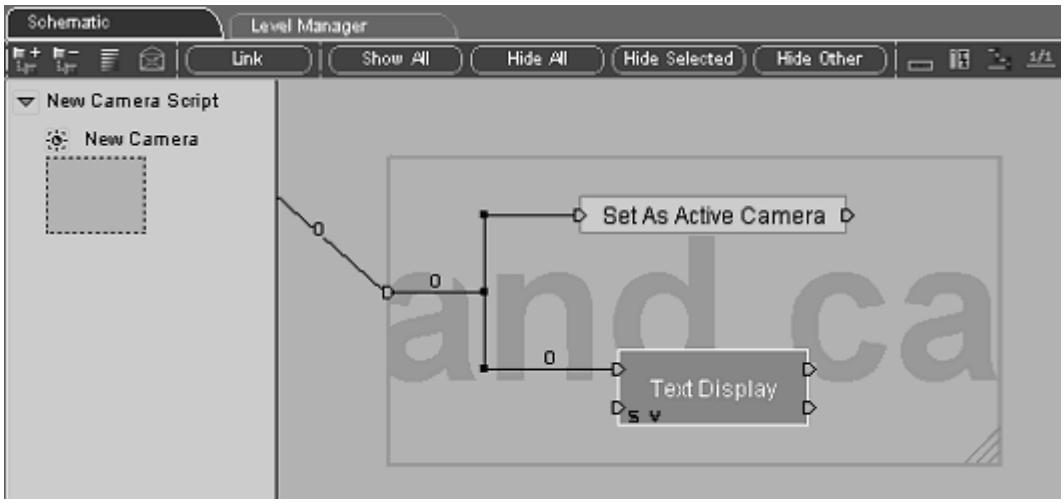


**Fig. 3.29** Un script comportemental avec Virtools Dev.

Virtools Dev semble au premier abord relativement simple d'utilisation. En effet, l'interface est très intuitive et, appliquer un comportement à un objet de la scène, est facile. Cependant, la création de scènes interactives complexes n'est pas aussi aisée et requiert une bonne connaissance du logiciel. Or, la pauvreté des documentations mises à disposition ne facilite pas cet apprentissage. La compréhension de l'utilité et du paramétrage de chaque boîte comportementale ne pose pas de problème particulier mais lorsqu'il s'agit de créer des comportements composés de plusieurs boîtes, il est souvent difficile de savoir précisément comment les combiner entre elles. Bien que plusieurs exemples existent, chacun d'entre eux nécessite une étude approfondie et une série de tests pour bien comprendre le fonctionnement de chaque script.

### 3.3.3 Interactions

Grâce à l'organisation sous forme d'arbre n-aire des éléments composant une scène 3D, un niveau suffisant a été atteint pour permettre, non seulement de décrire les objets, mais également, de manière indirecte, accélérer leur rendu. En effet, les primitives graphiques sont en général suffisantes pour décrire des objets complexes, et leur regroupement spatial est utilisé couramment afin d'éliminer les objets non visibles, grâce à la définition de boites ou de sphères englobantes. Mais, les interactions 3D sont toujours codées séparément et ne

font pas partie de la description de l'objet. Avant d'entrer dans les détails, définissons les différents termes employés.

Les animations sont des mécanismes qui permettent de faire vivre un objet automatiquement en changeant sa position, son orientation, son apparence ou n'importe lequel de ses attributs menant à une modification visuelle. On différencie par la suite, les animations déclenchées par un événement utilisateur (par une interaction) ou une animation automatique qui s'exécute une seule fois ou en continu, lorsque la scène est chargée pour la première fois.

Les interactions quant à elles, regroupent les actions que peut effectuer un utilisateur sur tout élément de la scène. Contrairement aux animations, elles ont ou non une répercussion visuelle immédiate. En effet, une interaction peut modifier une variable d'état de l'application sans pour autant modifier un objet. Néanmoins, l'action modifie l'application ; on peut donc la qualifier d'interaction. La majorité du temps, les interactions sont néanmoins visuelles. Avec celles-ci, l'utilisateur pourra agir en conséquence lorsqu'il verra une modification de l'état d'un programme. Les interactions sont caractérisées par un échange bilatéral entre l'homme et la machine pris en compte immédiatement. En aucun cas, les interactions sont caractérisées par l'utilisation d'une interface graphique. En effet, les interactions peuvent parfaitement être incluses dans une interface textuelle, dès lors que l'exécution du programme n'est pas séquentielle et automatique. Cependant, les interactions ont acquis toutes leurs forces représentatives avec les interfaces graphiques.

Nous abordons ici la manière de programmer ces interactions dans les interfaces 3D. L'implémentation usuelle par exécution d'un code associé à l'objet manipulé est également valable pour les interfaces 2D. La gestion par comportements, quant à elle, est commune à toute interface construite à partir base de *widgets*.

### Implémentation usuelle

Dans la plupart des applications 3D actuelles, qu'elles soient décrites ou programmées, la gestion des interactions s'effectue par association, puis par exécution d'un code propre à chacun des objets composant l'interface. Dans une application programmée, cela semble plutôt normal puisque l'ensemble de l'interface est spécifié par un langage de programmation éventuellement aidé par une bibliothèque de fonctions 3D. Dans une scène décrite, c'est-à-dire spécifiée dans un fichier dont le contenu est éditable et interprétable par une application qui en effectue le rendu, ce procédé d'adjoindre du code pour gérer des aspects que l'on ne peut pas décrire est souvent la seule alternative possible. Un mécanisme simple permet alors de déclencher l'exécution du code associé à un nœud lorsque l'utilisateur effectue une opération sur l'objet défini par ce nœud.

Dans le cas d'une interface 3D programmée, les principales étapes pour exécuter un code spécifique gérant les interactions sur un objet sont les suivantes :

1. récupération du type d'interaction (événement souris, clavier, ...),
2. si nécessaire, pour gérer l'interaction, récupération de l'objet sur lequel l'événement a eu lieu,
3. exécution du code associé à l'ensemble {événement, objet}.

Puisque les codes spécifiques aux comportements et les attributs donnant l'apparence des objets font partie du même programme exécutable (celui de l'application), tout code peut alors être exécuté, et modifier directement tous les objets. Il n'est donc pas nécessaire d'implémenter un mécanisme permettant de transmettre les événements de modification des attributs des objets. Les changements sont également automatiquement répercutés lors du prochain rendu de la scène car les commandes 3D, décrivant les objets, peuvent être également modifiées.

Les API 3D, qui fournissent les fonctions gérant toutes les opérations (de transformation, de projection, de coloriage, etc.), permettent généralement de gérer les données sous deux formes différentes : en mode immédiat ou en mode retenu. En mode immédiat, les commandes sont directement envoyées au pipeline 3D et traitées sans délai alors qu'en mode retenu, les commandes sont stockées dans une structure et envoyées d'une seule traite. Comme nous l'avons vu lors de la présentation des API 3D, selon les cas, les structures servant à retenir les commandes 3D sont différentes. En OpenGL, ce sont des listes de commandes séquentielles qui sont manipulées. Avec Java3D, c'est un arbre qui lie les différentes classes (d'objets, de vues, de regroupement) intervenant dans le rendu de la scène. Enfin, avec Direct3D, les deux structures peuvent être utilisées.

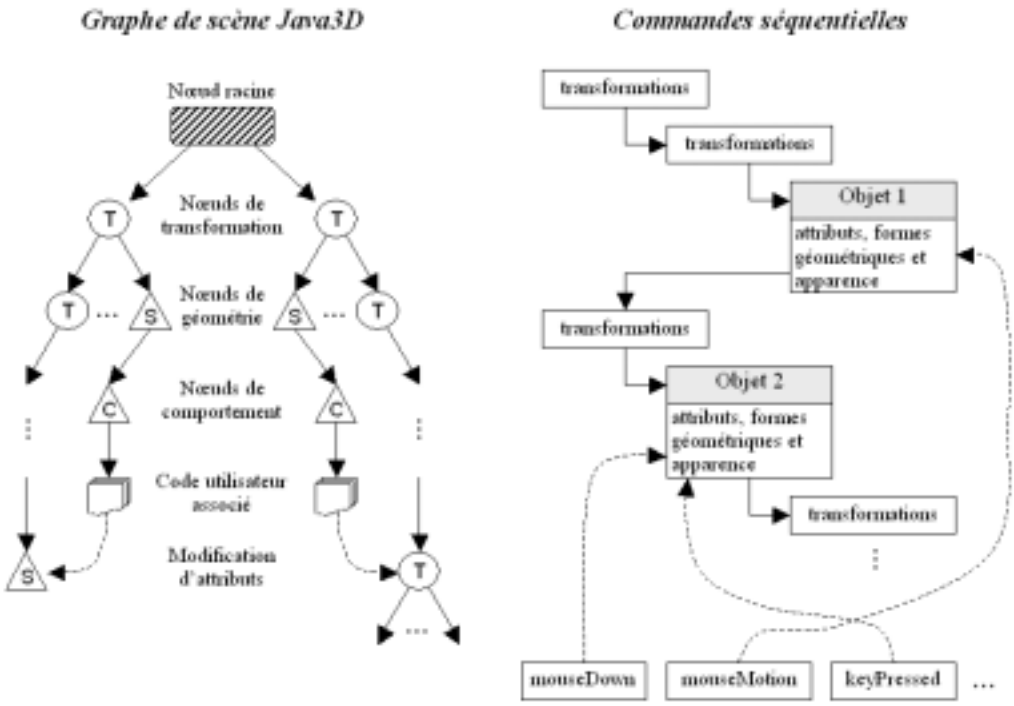


**Fig. 3.30** Associations des événement avec un code à exécuter.

Avec OpenGL et Direct3D, l'application peut modifier les commandes de rendu en fonction des interactions effectuées. En mode immédiat, les nouvelles commandes 3D, résultant d'une interaction, sont générées et envoyées au pipeline. En mode retenu, la structure sto-

ckant les commandes pour construire un objet est modifiée. Les événements sont récupérés comme pour une application 2D et opèrent les changements dans les structures de rendu comme cela est illustré pour les structures séquentielles dans le schéma à droite dans la figure 3.30 (mais le principe est le même pour modifier le graphe de scène de Direct3D).

En Java3D, les événements sont intégrés dans le graphe de scène (à gauche dans la figure 3.30). Des classes contenant un code utilisateur pour gérer les interactions peuvent être insérées par l'intermédiaire de la classe *Behavior*. Cette dernière peut être utilisée pour définir, par héritage, une classe dont le rôle est de gérer les événements d'un dispositif d'entrée particulier. Le code utilisateur peut, par l'intermédiaire d'un objet de cette classe, modifier les attributs des autres objets du graphe de scène. Les comportements, intégrés au graphe, apportent alors une clarté à la description des scènes. Ils sont attachés aux nœuds qui les ont activé et liés aux nœuds qu'ils modifient.

En VRML, langage dans lequel les données sont également organisées dans un graphe de scène, les comportements sont en partie décrits (voir la figure 3.31). Des nœuds capteurs (*Sensor*) permettent de récupérer des événements souris ou des événements de visibilité et de proximité. Ces événements peuvent être routés vers des nœuds d'interpolation (*Interpolator*) qui fournissent des valeurs interpolées à transmettre à un nœud de transformations (*Transform*) pour créer des animations. Dès lors que l'on souhaite effectuer des opérations plus complexes, pour changer les textures, par exemple, ou modifier une forme géométrique, un langage de script est nécessaire. Il s'intercale entre le capteur et le nœud modifié dans la chaîne de routages. Dans le fichier VRML, on ne pourra que décrire les animations automatiques ou déclenchées par un capteur. Toute opération sur le graphe de scène ne nécessitant pas de transitions données par les interpolateurs entraîne l'écriture d'un script.

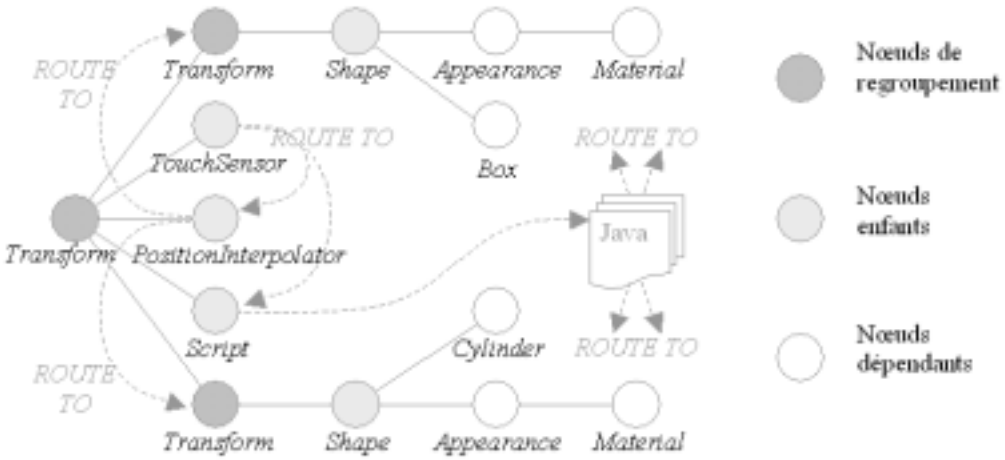


**Fig. 3.31** Exemple de graphe de scène VRML avec appel de script.

Tous les mécanismes précédents, permettant de programmer les comportements, nécessitent la connaissance d'un langage de programmation. Puisque ces langages sont compilés (mis

à part les scripts JavaScript avec VRML qui ne sont cependant pas aussi extensibles que ceux écrits en Java), trois étapes seront nécessaires pour mettre à jour le code :

1. édition et modification du code pour y intégrer une nouvelle fonctionnalité,
2. compilation pour produire l'exécutable après modification du code,
3. et exécution pour voir le résultat de la modification.

L'approche en spirale ou intuitive est donc difficile à mettre en œuvre avec les codes compilés. Au contraire, un langage interprété de type Alice [Conway00] permet de supprimer l'étape de compilation. L'auteur d'une application 3D peut ainsi évaluer en un temps donné, beaucoup plus de modèles d'interaction et d'options de rendu pour une interface. Enfin, le temps d'apprentissage d'un tel langage est beaucoup plus court que pour un langage de programmation. La différence est d'autant plus importante si le langage compilé nécessite l'apprentissage de concepts tels que celui de la programmation par objets. Or, comme cela est dit dans l'article présentant les leçons tirées des manipulations faites par des novices avec Alice [Conway94], un nouveau type d'auteurs tels que les artistes ou les designers sont intéressés par la 3D. Et ces créateurs ne disposent pas, la plupart du temps, des connaissances mathématiques et informatiques nécessaires pour créer une scène 3D.

Pour embrasser l'auditorat le plus large possible, Alice manipule des notions proches du langage naturel (tourner, bouger) au lieu d'objets mathématiques (matrices, équations). Ce langage n'est qu'un langage de simulation ; le moteur 3D associé, DIVER gère la représentation interne des objets et leur rendu [Gossweiler93]. Ces deux moteurs (de simulation et de rendu 3D) sont découplés et échangent des messages pour communiquer. Cela permet, comme l'a montré [Shaw92], d'avoir des fréquences différentes entre le rendu de deux trames successives et le calcul de deux états de la simulation. Bien que le langage Alice ait été développé afin d'offrir un vocabulaire simple pour spécifier les comportements, sa syntaxe est tout de même très fortement inspirée par les langages orientés objet. Il s'appuie en effet sur le langage Python qui est lui-même proche du langage objet C++. Il est alors tout à fait compréhensible que certains non-informaticiens restent tout de même démunis face à ce langage, non pas à cause de la syntaxe des interactions mais à cause de la syntaxe en Python permettant de les spécifier.

Même une construction visuelle des interactions, avec des outils tels que Cult3D Designer ou Virtools Dev, nécessite la connaissance des opérations de transformations et une certaine habitude dans la manipulation des graphes. La compilation idéale des différents projets précédents semblerait être un système d'édition des interactions, comme celui proposé par Cult3D dans lequel le langage associé aux transitions, serait du type de celui d'Alice. Ce système devrait alors générer un automate simple associé au graphe de scène dans le même fichier. Il serait ainsi éditable par les informaticiens qui préfèrent souvent avoir le contrôle des procédés de fabrication du code.

### Les comportements

Nous parlerons plus loin dans ce rapport des comportements (voir la section 5.3.1 page 200). Ceux-ci représentent la nouvelle manière de programmer les interactions avec les objets. Les comportements englobent les animations et les interactions vues précédemment. Ils

définissent toutes les méthodes permettant de modifier un attribut de l'objet auquel ils sont attachés.

## 3.4 Evolution du matériel

Nous avons déjà examiné convenablement dans la section 3.3.1 page 88, les différents éléments de la couche matérielle intervenant dans l'obtention des images de synthèses. Cependant, il existe d'autres éléments importants intervenants dans le domaine de la 3D temps réel. Nous présentons succinctement quelques dispositifs de pointage disposant des six degrés de liberté, et convenant donc mieux que les *widgets* de navigation pour interagir avec une scène.

Pour parfaire la sensation d'immersion, il est également important de mettre à contribution l'ensemble des sens et non pas uniquement la vue. En attendant l'évolution et la démocratisation des dispositifs de synthèse des odeurs, d'autres sens peuvent être stimulés. Parmi ceux-ci, le toucher par l'intermédiaire des périphériques à retour d'efforts, et l'ouïe grâce à la synthèse sonore 3D.

### 3.4.1 Dispositifs d'entrée/sortie

La plupart des opérations d'entrée/sortie à partir des périphériques standards, se font en deux dimensions : les entrées par des dispositifs 2D, comme la souris ou le joystick qui comptent deux degrés de liberté (deux translations), et les sorties sur imprimante ou moniteur qui ne peuvent que coucher des images à plat. Des périphériques spécialisés existent cependant, pour interagir réellement en 3D ou pour obtenir une sortie 3D. Ces périphériques 3D se sont multipliés et l'on compte désormais parmi eux, les gants de réalité virtuelle et les souris 3D offrant les six degrés de liberté (trois translations et trois rotations). La plupart des dispositifs de visualisation 3D sont cependant encore réservés aux domaines de la recherche, des technologies de pointe ou des simulateurs militaires. C'est le cas notamment des systèmes de représentation holographique comme le MIT holovideo display [Watlington95], des caves équipées d'écrans offrant une vue panoramique, des *workbenchs* [Obeysekare96] où les images sont projetées sur une table et des imprimantes 3D créant des volumes.

#### Périphériques de sortie

Toutes les configurations actuelles bien que ne possédant pas ces périphériques, embarquent cependant des possibilités 3D. 3D étant l'abréviation de trois dimensions, comment expliquer que sur un écran en deux dimensions nous puissions obtenir une sortie 3D ? C'est sur ce point que la définition de la 3D pour l'informatique est quelque peu différente de celle du monde physique puisqu'elle intègre, en plus de la manipulation d'objets 3D, la manipulation d'images 2D (projections à l'écran d'objets décrits dans l'espace) donnant l'impression de volume. Les techniques utilisées pour créer cette impression sont décrites en détails dans de nombreux ouvrages dont [Foley90, Watt93].

Mais précisons, dés à présent, que les méthodes de création et de visualisation d'un contenu 3D utilisent toutes ces techniques et ne permettent donc que de donner à l'utilisateur l'illusion qu'il évolue dans un "monde 3D". La plus grande immersion, à un prix abordable, est obtenue avec les casques de réalité virtuelle ou les lunettes, grâce à leur rendu stéréoscopique. Dans les deux cas, deux images sont calculées, une par œil. La seule différence entre ces deux images provient d'une légère translation de la prise de vue sur la scène, pour simuler l'écartement des yeux et donc la vision depuis chacun des yeux. Pour le casque, les deux images sont affichées simultanément sur deux écrans à cristaux liquides différents (un par œil). Avec les lunettes, le rendu est effectué sur le moniteur. L'effet stéréoscopique est obtenu par l'affichage alterné des images calculées. Pour percevoir le relief, les lunettes servent d'obturateur. Elles sont, en effet, synchronisées avec l'affichage et lorsque l'image pour un œil est affichée, l'autre œil est obturé pour ne pas la voir. Une fréquence double - 50 images par seconde - est nécessaire pour satisfaire la tolérance individuelle de chacun des yeux.

La principale différence entre le monde physique que nous percevons en 3D ou la représentation holographique et la fausse 3D plus communément utilisée, est l'immobilité des objets lorsque nous bougeons la tête en face de l'écran. Ces systèmes, à l'exception des casques avec capteurs de position, ne permettent pas, par une translation de la tête ou du regard, de découvrir plus d'informations sur le volume d'un objet. Cependant, l'illusion est suffisante pour donner à l'utilisateur le sentiment d'être plongé dans un univers 3D ; d'autant plus s'il utilise un environnement immersif visuel et sonore.

Cependant, cette course à l'immersion soulève un problème majeur quant à la créativité liée à l'utilisation de la 3D. En effet, la plupart des applications 3D tentent de recréer un monde virtuel proche du monde réel. Bien que cela permette de mettre en valeur les expériences acquises par les utilisateurs, ces interfaces ne favorisent pas la création. Nous reviendrons plus loin dans ce rapport sur ce problème mais pour le clarifier dès à présent, nous pouvons utiliser une comparaison. La 3D telle qu'elle est utilisée aujourd'hui est dans une phase technique de reproduction du réel, tout comme le cinéma, qui à ses débuts, ne montrait que des clichés animés de scènes quotidiennes. Alors que le cinéma a évolué en exploitant les *travellings*, les contre-champs, la 3D, de la même manière, doit maintenant explorer de nouvelles métaphores différentes de celles issues de la réalité.

## Dispositifs de pointage

Les techniques d'interaction avec une scène 3D autorisant l'utilisateur à se déplacer sur les six degrés de liberté, à examiner les objets et en plus à exécuter les actions associées avec un objet, sont d'une grande importance dans l'élaboration d'une interface 3D. A la lecture de différents articles discutant des interactions, nous nous rendons compte qu'il existe un étonnant décalage entre les techniques utilisées dans les applications 3D actuelles et celles qui pourraient être utilisées. Comme pour les interfaces 2D "modernes" que nous utilisons depuis leur création dans les années 70, le pointeur le plus couramment utilisé, la souris 2D, n'a pas fait non plus l'objet d'un extraordinaire renouveau et tend à prouver que l'évolution de l'une est liée à celle de l'autre.

### *Interactions 3D avec des dispositifs 2D*

La souris s'est donc imposée pour interagir avec une scène 3D. Bien que celle-ci ne compte que deux degrés de liberté (elle permet uniquement de faire deux translations dans le plan horizontal), elle compte néanmoins des avantages. Les utilisateurs la manie en effet avec maîtrise et il est ainsi possible de profiter de cette connaissance. Toutes les opérations de sélection et d'interaction avec les objets 3D pourront (et devrons) en tirer partie. Des études ont sur ce point été menées sur la manipulation directe d'objets 3D à partir d'une souris 2D [Chen88, Emmerik90].

A l'heure actuelle, pour les applications 3D grand public, il existe deux modes d'interaction différents. Le premier demande l'utilisation conjointe de la souris et du clavier. Certaines touches effectuent les déplacements sur les différents axes ou des actions. Les rotations sont calculées à partir des mouvements de la souris avec un bouton appuyé. Les autres boutons de la souris restent alors disponibles pour l'implémentation de la sélection ou d'actions. Le deuxième mode utilise la souris et des *widgets*. Les *widgets* viennent en surimpression ou à côté de l'objet sélectionné et proposent des actions sur les objets. Par exemple, dans les navigateurs VRML, des images représentent le type de déplacement que l'on peut effectuer avec la caméra. Par un clic de souris sur un de ces composants, un mode de navigation est sélectionné. A la suite de ce clic, les déplacements de la souris dans la scène sont interprétés pour effectuer la navigation sélectionnée.

Certaines méthodes ont cependant été étudiées et implémentées pour permettre une navigation relativement complète en utilisant uniquement la souris. Même si elles ne sont que des alternatives comparées à l'utilisation d'un pointeur 3D, elles tirent néanmoins partie de la présence des souris 2D sur tout ordinateur. Parmi celles-ci, nous en avons relevées deux particulièrement originales.

R. Zeleznik et A. Forsberg ont présenté une manière de contrôler une caméra virtuelle 3D sur les six degrés de liberté, à l'aide d'une souris 2D et d'un seul de ses boutons ; sans avoir besoin d'un clavier ou de *widgets* 2D [Zeleznik99]. UniCam, permet en effet de garder les autres boutons de la souris libres pour les opérations spécifiques d'une application, contrairement à la plupart des autres systèmes de navigation dans un espace 3D. Prises individuellement, les huit techniques de navigation implémentées ne sont que de modestes adaptations de techniques bien connues (translations, rotations, sauts, etc.). L'originalité d'UniCam est de permettre une sélection de la métaphore de navigation désirée par de simples transitions gestuelles[13]. Il est ainsi inutile d'aller cliquer sur un bouton comme c'est le cas avec les *widgets* de navigation.

L'article de R. Zeleznik, A. Forsberg et S. Strauss [Zeleznik97] explore différentes techniques d'interactions bi-manuelles dans une applications 3D. D'après les résultats obtenus pour les applications 2D, les auteurs pensent que les interactions avec les deux mains permettent de créer des interfaces plus efficaces et plus fluides, particulièrement pour les tâches dépendantes d'un contexte ou qui ont beaucoup de degrés de liberté. Et, ils formulent l'hypothèse que l'interaction à deux mains est plus efficace qu'à une main, hypothèse qu'avaient vérifiés Buxton et Myers [Buxton86]. Ces tâches nécessitant les deux mains apparaissent

[13] C'est ce système de sélection des opérations de navigation que nous avons choisi d'implémenter dans notre navigateur VRML [Topol01].

souvent dans les applications 3D et sont généralement exécutées par une série d'opérations séquentielles, exigeant moins de degrés de liberté. Les interactions avec les deux mains permettent, en théorie, d'effectuer les mêmes tâches en utilisant la moitié d'étapes séquentielles, puisque deux opérations élémentaires peuvent être effectuées en même temps. De plus, de nombreuses interactions à deux mains sont plus facilement compréhensibles et utilisables, puisqu'elles ressemblent voire correspondent à celles que nous exécutons dans le monde physique[14].

### *Interactions 3D avec des dispositifs 3D*

L'avenir appartient, cependant, aux nouveaux dispositifs de pointage ou capteurs dont les mouvements peuvent être plus naturellement associés à la navigation ou à la manipulation d'objets, dans un espace 3D. Les nouvelles méthodes qui émergent ont été classées en deux catégories en fonction du type de gestes les contrôlant [Baecker95]. Pour chacune d'elles nous donnons quelques exemples :

- Non haptique[15] (sans sensation de contact physique avec les objets virtuels) :
  - Caméra : utilisation d'une caméra vidéo dont les images capturées sont segmentées pour extraire les informations importantes. Cette méthode demande une grande puissance de calcul à cause des traitements en temps réel des images. Elle est également dépendante des paramètres de l'environnement comme les variations de lumières.
  - Commandes vocales.
  - Sondage de champ électrique [Smith98] : mesure de la charge accumulée entre deux groupes d'électrodes qui varie lorsqu'il y a présence d'un objet entre les deux.
  - Champ électromagnétique [Fuchs99] : utilisation de deux groupes de trois bobines. Pour l'émetteur, trois bobines propagent un champ électromagnétique dans chacune des trois directions orthogonales (x, y, z). Les 3 bobines du récepteur, également orthogonales, recueillent ces flux en fonction de leur position et transmettent les coordonnées du récepteur. Cette méthode bien que très fiable et ayant une bonne fréquence de mesures reste coûteuse et souffre des possibles interférences engendrées par tout objet métallique.
  - Ultrason [Bibes97, Fléty97, Fuchs99] : le principe est le même que précédemment mais il utilise des ultrasons. Les principaux problèmes de cette méthode sont le délai de propagation des ondes sonores et les interférences dues aux réflexions.
  - Infrarouge [Bibes97] : un émetteur envoie des rayons infrarouges à un récepteur qui mesure l'angle d'arrivée et la variation de l'intensité des rayons. Cette technique est inefficace lorsqu'un corps se trouve entre l'émetteur et le récepteur. L'angle d'arrivée des rayons ne doit pas non plus excéder un angle de 40 degrés.
  - Capteurs à effets Hall : calcul de la tension aux bornes d'un semi-conducteur récepteur lorsque celui-ci est soumis à un champs électrique et magnétique dû à l'alimentation du capteur.
  - Inclinomètres : indication de la position angulaire par rapport à la verticale. Lorsque le capteur est en mouvement, cette technique indique l'accélération.
  - Compas : exploite l'effet Hall à partir du champ magnétique terrestre.

[14] Par exemple, pour éplucher une pomme, on l'oriente pour y accéder plus facilement avec le couteau.
[15] Du grec *haptein* signifiant toucher.

  - Gyroscopes et gyromètres : mesure de la vitesse angulaire d'un système en rotation autour d'un axe en utilisant le phénomène de la force de Coriolis.
  - Capteurs de pression mécanique : capteurs dont la résistance électrique diminue lorsqu'on leur applique une pression.
  - Capteurs de flexion à encre conductrice [BURDEA93] : la flexion exercée augmente la distance entre les particules de carbone des deux couches d'encres conductrices, augmentant ainsi la résistivité du capteur.
  - Mesure de signaux biologiques [PICARD97] : utilisation de la tension musculaire, du rythme cardiaque, de la position des yeux, des ondes cérébrales, de la pression sanguine, etc. pour contrôler un système.
- Haptique (les interfaces avec retour tactile et retour de force ou d'effort) : joystick, souris, stylet, gant.

Les principaux périphériques d'entrée 3D que l'on trouve dans le commerce se présentent sous la forme de *trackball* 3D ou de capteurs que l'on fixe sur la main ou la tête. Ils gèrent tous au moins 3 degrés de liberté (pour les trois translations) et certains mesurent les rotations, donc 3 degrés de plus. Les capteurs les plus perfectionnés peuvent en compter plus. Le gant par exemple gère la position de la main et la flexion de chacun des doigts. Ces périphériques 3D ont été classés par C. Dumas [DUMAS99B].

Les périphériques isométriques, comme les souris 3D ou les *trackballs* 2D, possèdent une résistance infinie, c'est-à-dire qu'ils sont immobiles. Ils mesurent les mouvements en fonction des forces appliquées. Leur défaut principal est la dé-corrélation entre le mouvement de la main (quasi nul) et celui du pointeur. Le manque de retour de force gène également pour les tâches de sélection par exemple.

Les périphériques isotoniques possèdent eux une résistance nulle (*i.e.* ils se déplacent avec la main). En 2D, c'est le cas des souris et en 3D celui des gants ou des systèmes à ultrasons. Leur principal défaut est la fatigue de la main après une utilisation prolongée.

Les périphériques élastiques sont à mi-chemin entre les deux précédents. Ils retournent automatiquement à une position stable dés lors que l'on cesse d'y appliquer une force. Ils sont réputés pour leur utilisation plus intuitive grâce aux retours de force.

Le débat n'est toujours pas clos sur le type de périphériques 3D d'entrée le plus adapté à la navigation ou à la sélection d'objets dans l'espace. Chaque périphérique compte des avantages pour certaines manipulations mais des inconvénients pour d'autres [DUMAS99B, ZHAI94]. Il s'avère que les périphériques isométriques sont préférés pour la navigation et les périphériques isotoniques pour la sélection. Les expérimentations mises en œuvre par C. Dumas prouvent un léger avantage pour un périphérique isotonique lors d'une tâche de sélection. Cette étude a comparé le temps et le nombre de clics émis pour sélectionner un objet dans l'espace, en fonction de différents paramètres comme le niveau d'expertise de l'utilisateur, le type du périphérique (isotonique ou isométrique) et la présence d'ombres sous les objets. La combinaison optimale est obtenue avec les experts utilisant le périphérique isotonique pour sélectionner des objets ombrés. Cette a également permis de valider l'utilité des ombres pour fournir une information de profondeur.

### 3.4.2 Spatialisation du son

Comme nous avons pu le voir précédemment dans les différents projets et techniques, les graphismes 3D sont reconnus, depuis quelques années, comme pouvant offrir une bonne représentation des informations. Ces nombreux projets insistent sur le fait que les données peuvent être perçues par le système perceptif plutôt que de devoir être analysées par le système cognitif. Cependant, les propriétés graphiques des objets ne sont pas les seules à pouvoir être spatialisées et à bénéficier de l'apport de nos capacités perceptives. Les informations sonores peuvent aussi aider à comprendre certaines informations et animations 3D [Herder97].

De plus en plus, nous voyons apparaître des technologies permettant d'évoluer dans un univers sonore 3D. Les jeux vidéos, qui font une excellente promotion du son 3D comme ils l'ont fait auparavant pour les graphismes 3D, s'appuient sur plusieurs API spécialisées. Parmi les plus utilisées, DirectSound de Microsoft, s'intègre particulièrement bien à la 3D temps-réel puisque cette bibliothèque fait partie avec Direct3D de l'ensemble DirectX. D'autres alternatives sont cependant possibles : EAX et OpenAL en sont des exemples. Ces dernières ont d'ailleurs un avantage particulier sur DirectSound puisqu'elles sont multi-plateformes.

D'autres domaines de l'industrie audiovisuelle accordent une grande importance à la restitution sonore 3D. Les premières disciplines à s'y être intéressées, dans les années 70, sont l'informatique musicale et l'acoustique architecturale. Mais l'intensification de ces recherches est nette depuis l'apparition des systèmes multimédias, des jeux vidéos et plus généralement de la course à l'immersion, dans les systèmes de réalité virtuelle. Non seulement la spatialisation sonore augmente la sensation d'immersion [Tsingos97], mais elle développe aussi la perception des informations et donc la lisibilité des interfaces. Les sources sonores virtuelles peuvent, par exemple, palier l'absence d'indices visuels. L'exemple le plus fréquent se trouve probablement dans les jeux vidéos où le joueur peut entendre un autre joueur lui tirer dessus et ainsi deviner sa position. Mais on pourra citer aussi l'exemple des "oreilles d'or". Ce sont des agents de centrale électrique, capables de détecter les problèmes d'après les bruits inhabituels et qui pourraient exercer leur talent devant une console virtuelle grâce aux techniques de spatialisation sonore.

S. Natkin [Natkin00] envisage d'autres applications originales, comme la réalité augmentée sonore permettant d'évoluer dans un lieu réel augmenté de sons de synthèse. Dans ce cas particulier de superposition des deux mondes réel et virtuel, plusieurs problèmes se posent avant de pouvoir transmettre un son spatialisé aux utilisateurs. On devra étudier les aspects suivants :
- la représentation du monde virtuel,
- la localisation des utilisateurs dans le monde réel et leur correspondance dans le monde virtuel,
- l'envoi du son de synthèse,
- la distribution des calculs coûteux, selon le nombre d'utilisateurs et la forme du monde virtuel.

C'est dans la perspective du premier point que nous avons étudié l'utilisation de VRML pour représenter des scènes sonores complexes (voir section 4.6 page 180). Cette étude a

mené à quelques améliorations de la description des sons spatialisés dans VRML. Nos extensions sonores permettent de décrire des mondes virtuels plus riches, et donc de naviguer à travers un contenu en ligne plus riche qu'avec le standard VRML.

## Principes généraux

On entend généralement par spatialisation sonore, la localisation et la perception de sons dans l'espace, c'est-à-dire pouvoir déterminer la source des sons. Cependant, cette localisation n'est pas suffisante pour obtenir un effet réaliste car d'autres phénomènes entrent en jeu lors de la propagation sonore. En particulier, l'effet de salle[16] détermine les réflexions et les réfractions des sons. La figure 3.32 illustre l'échogramme schématique de la réponse d'une salle à un son. On y voit que, sans la prise en compte de l'effet de salle, seul le son direct, une infime partie de ce que l'on entend en réalité, est perçu.

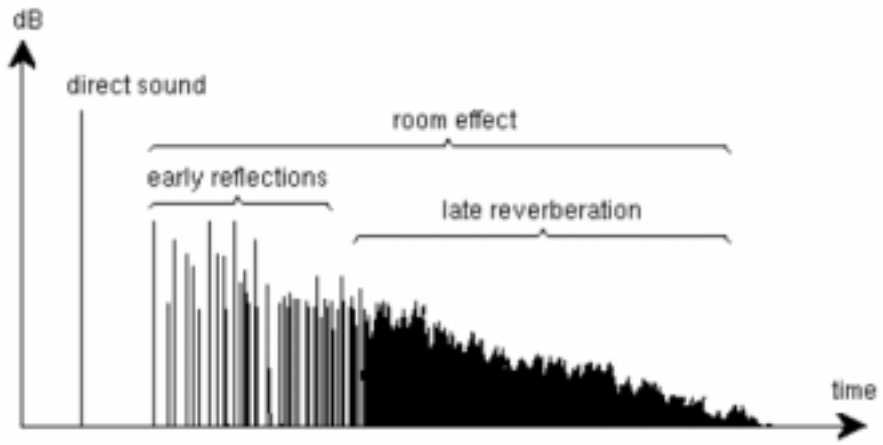


**Fig. 3.32** Un échogramme schématique prenant en compte l'effet de salle.

La spatialisation de sources sonores virtuelles peut être obtenue de deux manières principales, illustrées dans la figure 3.33 : par enregistrement d'une scène sonore réelle ou par synthèse d'une scène sonore virtuelle. La première, à gauche dans la figure 3.33, nécessite une mise en place de microphones par des spécialistes aux endroits où se trouveront virtuellement les auditeurs. Cette méthode permet d'enregistrer toutes les informations relatives à la localisation des sources mais restreint les possibilités de modification des sources sonores virtuelles correspondantes. Puisque les sons enregistrés contiennent déjà toutes les composantes de l'effet de salle, il est, par exemple, difficile de leur appliquer un effet de salle différent de celui du lieu réel dans lequel a été effectué l'enregistrement.

La deuxième manière d'obtenir des sources sonores virtuelles spatialisées (à droite dans la figure 3.33) est d'utiliser des algorithmes de synthèse. Avec cette technique, les sons manipulés sont élémentaires et monophoniques. On peut ainsi leur appliquer toutes sortes

[16] On regroupe sous le terme "effet de salle" toutes les caractéristiques acoustiques d'un lieu.

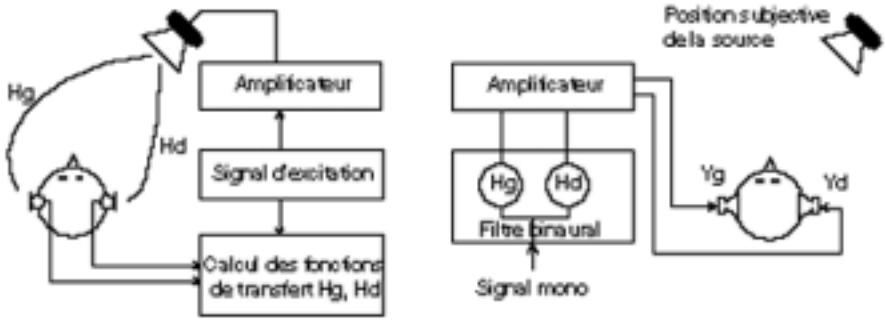


**Fig. 3.33** Les deux méthodes de spatialisation sonore.

de filtres parmi lesquels ceux reconstruisant leur localisation et ceux prenant en compte l'effet de salle de leur milieu. Avec un seul enregistrement de son il est donc possible de calculer une multitude de sons virtuels spatialisés. Il suffira pour cela de fournir aux algorithmes la localisation et l'orientation de l'auditeur, celles des sources sonores, leur directivité et l'effet de salle de la scène virtuelle. Lorsque les algorithmes peuvent traiter tous les sons en temps réel, on pourra alors se déplacer interactivement dans la scène peuplée de sources sonores mobiles.

## Les deux techniques de synthèses sonores 3D

Il existe deux techniques différentes pour calculer la spatialisation de sources sonores virtuelles et leurs interactions avec la géométrie de l'univers 3D. Plus exactement, deux méthodes sont disponibles pour calculer l'influence de l'effet de salle sur les sons. Pour chacune d'elles, les paramètres à prendre en compte et les algorithmes à mettre en place sont totalement dissemblables. S. Natkin a décrit ces différences fondamentales et les a rapprochées des différences qui existent entre le lancé de rayons et la 3D par projection de facettes [Natkin98]. Nous présentons ici ces deux principaux modèles pour spatialiser les sources sonores.

### *Le modèle physique*

Ce premier modèle [Savioja97, Funkhouser98] ressemble le plus à la propagation des sons dans le monde réel. Il exploite les propriétés de réflexion et de réfraction des objets pour calculer le trajet des ondes sonores. Avec cette technique, les sons perçus par l'utilisateur dépendront également de la géométrie de la pièce et de la distribution d'énergie qui la caractérise. Les sons sont calculés en "tirant" des ondes depuis chacune des sources sonores jusqu'à l'oreille de l'utilisateur, après de possibles (et probables) réflexions et réfractions (voir figure 3.34). Cette méthode est fortement inspirée de la synthèse d'images par lancé de rayons (ou *raytracing*).

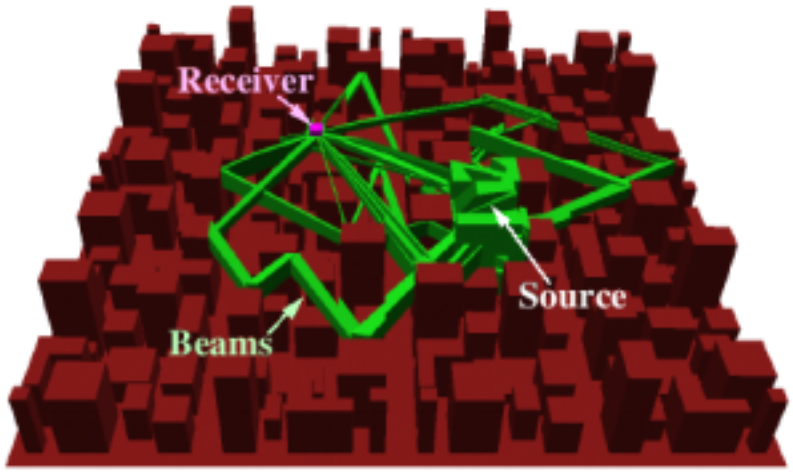


**Fig. 3.34** La synthèse sonore avec le modèle physique [Funkhouser98].

Le problème avec ce modèle physique est qu'il nécessite de nombreux calculs difficilement réalisables en temps réel. Des algorithmes ont été développés pour accélérer ce type de rendu et le rendre interactif. En particulier, l'observation des lois physiques statistiques de distribution spatio-temporelle de l'énergie dans des espaces clos, a permis de simplifier considérablement le modèle de base[17]. Bien qu'accélérant le rendu sonore, ces méthodes ne laissent néanmoins aucune place au rendu graphique. La synthèse audio et visuelle est par conséquent impossible, sur les machines actuelles, avec ce modèle physique.

### *Le modèle psycho-acoustique*

Dans ce deuxième modèle de synthèse des sources sonores virtuelles [Jot97], le calcul des sons est indépendant de la géométrie de la scène. On utilise uniquement des attributs perceptifs (comme la réverbérance de la salle ou l'humidité de l'air ambiant) pour calculer la somme des réflexions et des réverbérations dans le monde virtuel. Des expérimentations ont été réalisées pour comparer l'acoustique obtenue par cette méthode de synthèse avec celle du véritable lieu [Warusfel95, Marin96]. Elles ont montré qu'il est possible d'avoir une synthèse correcte avec peu de paramètres globaux, comme ceux de la figure 3.35 montrant une copie d'écran du Spatialisateur de l'IRCAM [Jot95].

L'intérêt de cette méthode de synthèse psycho-acoustique, outre son aspect temps réel et réaliste, est de pouvoir simuler des environnements qui ne pourraient pas être réalisables dans le monde réel ou obtenus avec un modèle physique. Par exemple, la scène 3D peut être une petite pièce rectangulaire dans laquelle on utiliserait des attributs d'effet de salle d'une cathédrale.

On voit apparaître par ailleurs de nombreuses bibliothèques de fonctions et des logiciels qui permettent de manipuler facilement ce type de synthèse sonore. On citera notamment

[17] On notera encore la ressemblance entre ce modèle et les algorithmes de radiosité utilisés en synthèse d'images.

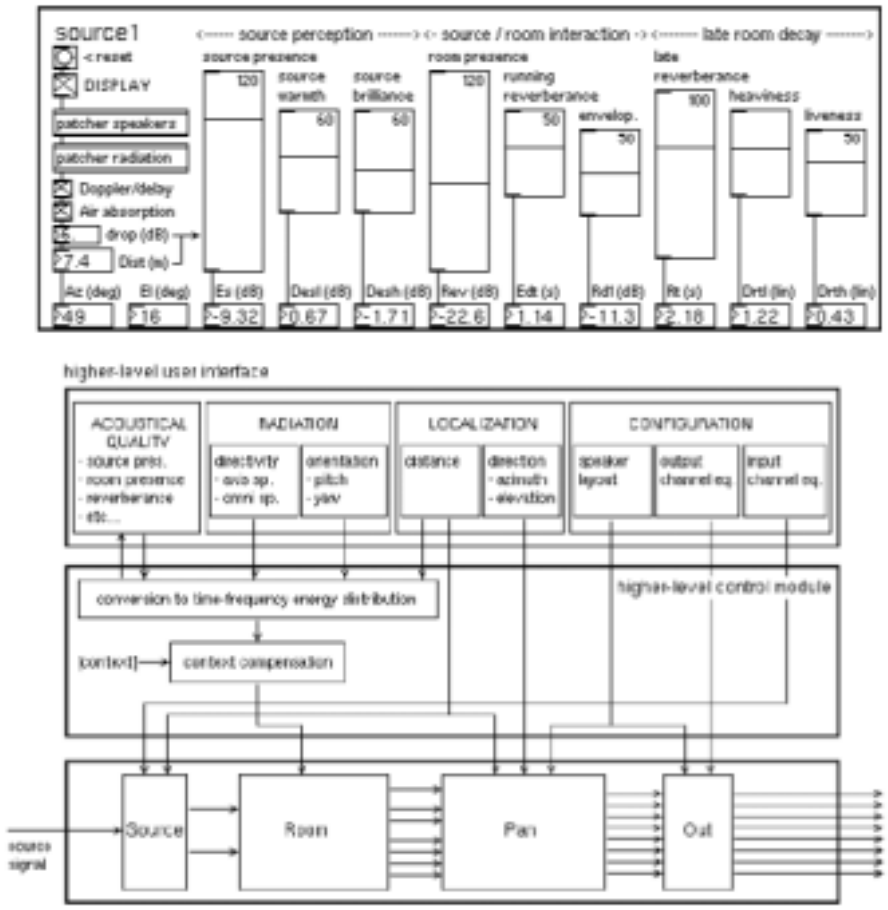


**Fig. 3.35** Les paramêtres de spatialisation du Spat de l'Ircam [Jot95].

EAX, EAX Manager, Eagle de Creative Labs mais également OpenAL. Quant à l'inclusion des attributs perceptifs sonores dans une description de scène 3D, MPEG4 semble le projet plus avancé.

# Chapitre 4

# *Expérimentations*

**Contenu du chapitre**

*Dans ce chapitre, sont présentées les diverses expérimentations que nous avons faites. Les premières s'attachent à la visualisation et à l'accès en 3D des bibliothèques numériques. Les secondes s'attèlent à la création d'un "environnement applicatif" (ou bureau) 3D autorisant l'exécution des applications 2D et 3D. Enfin, les troisièmes se tournent vers la modélisation et la restitution sonores en 3D.*

*Ce chapitre est organisé de la manière suivante :*


- Section 4.1 – Introduction du chapitre – p. 130 : *Présentation générale de la problématique et la manière de la traiter.*
- Section 4.3 – Les plateformes d'expérimentation – p. 133 : *Présentation des logiciels ayant permis d'effectuer les différentes expérimentations.*
- Section 4.4 – Bibliothèques numériques – p. 142 : *Nos expérimentations sur les bibliothèques numériques.*
- Section 4.5 – Bureau 3D – p. 174 : *Présentation de notre gestionnaire de programmes en 3D sous XWindow.*
- Section 4.6 – Son spatialisé – p. 180 : *Extension du modèle sonore de VRML par la prise en charge des paramètres nécessaires à la synthèse psycho-acoustique.*

## 4.1 Introduction du chapitre

Proposer des solutions à certains problèmes particuliers pour ensuite les généraliser, peut paraître peu commune. Généralement, à partir d'un problème donné, des hypothèses sont émises afin de le résoudre, pour ensuite vérifier qu'elles conviennent bien après validation.

Cette manière de faire nous a été imposée dès lors que nous avons choisi l'outil principal sur lequel nous avons travaillé. En effet, le langage sur lequel nous nous appuyons pour décrire et mettre en ligne nos scènes 3D est, jusqu'à maintenant, VRML. Or, ce langage est généraliste dans le sens où il permet de décrire une scène 3D sans s'attacher à son utilité. On peut simplement le voir comme un format de fichier textuel très proche dans ses fonctionnalités, des formats de fichier DXF d'Autocad ou 3DS de 3D Studio Max puisqu'il permet de fournir une description des caractéristiques d'objets 3D. Le langage définit donc bien une syntaxe mais pas une sémantique. Le sens, l'utilité donnés aux objets ne sont absolument pas inclus. Afin de conserver cette caractéristique généraliste du langage VRML, nous avons donc essayé de proposer des extensions détachées d'un domaine ou d'une application particulière.

Néanmoins, nos supports de réflexion ont été des applications particulières comme le bureau 3D ou les bibliothèques numériques. Et celles-ci nous ont permis d'extraire des interactions propres mais toutefois généralisables. Nous avons, par conséquent, pu créer des comportements génériques utilisables dans l'une de nos deux applications. Ainsi, nous pouvons voir ce travail comme une extension du langage VRML afin qu'il réponde à nos applications sans pour autant en changer sa nature ouverte. Ces modifications restent suffisamment générales pour être utiles et utilisables dans d'autres contextes que les bibliothèques numériques.

Nous présentons ici les travaux réalisés durant trois années de thèses. Ces différents projets peuvent, au premier abord, paraître indépendants et donc former un ensemble hétérogène. Néanmoins, bien qu'étant apparemment sans lien, ils répondent tous à la même problématique : la 3D peut-elle être utilisée pour la manipulation de données multimédias ? La 3D descriptive nous permet-elle, à l'heure actuelle, de spécifier des applications 3D ? Les propositions générales que nous présentons dans le dernier chapitre de ce rapport sont issues de ces différentes expérimentations.

Dans un premier temps, nous décrivons dans ce chapitre le contexte dans lequel nous avons développé notre plateforme d'expérimentation et les outils sur lesquels elle s'appuie. A la suite de cette partie dans laquelle nous expliquons le choix de nos outils, nous présentons nos différents projets et expérimentations ainsi que les critiques et remarques que nous pouvons en faire.

## 4.2 La toute première étude

Nous avons débuté notre étude de VRML en 1996, par la réalisation d'un simulateur urbain. L'idée principale de ce simulateur et de son éditeur d'espace urbain associé, est de permettre la modélisation 3D d'un modèle urbain avec les outils d'étude habituels utilisés par un architecte, à savoir feutres et calques. L'éditeur permet de créer rapidement une

scène VRML correspondant au projet urbain. Une promenade virtuelle à l'intérieur de cette scène audio-visuelle est ensuite possible pour évaluer le projet. L'objectif était donc de proposer un outil ergonomique permettant de prototyper rapidement une maquette interactive 3D d'un projet.

VRML nous a semblé d'emblée le format de fichier idéal pour générer les scènes et permettre aux architectes de communiquer leurs projets sur Internet et aux utilisateurs de se promener en temps-réel dans le nouvel espace urbain (à l'aide d'un *plugin*). Ce format de fichier s'est imposé de suite comme le format de sauvegarde des scènes générées. Notre simulateur urbain peut ainsi être vu comme un modeleur VRML spécialisé.

L'objectif d'un urbaniste est d'étudier et de créer un espace urbain dans un site existant par ajout, suppression ou modification du tissu urbain, bâti, réseau, équipements, espaces verts. La modélisation d'un quartier est donc basée sur l'existant : le plan des courbes de niveaux du terrain, du bâti et, par exemple, d'emplacements d'espaces verts. Les paramètres tels que la hauteur de bâti, le type d'espace vert etc., seront associés en cliquant dans la zone du plan représentant l'objet à éditer. Les aspects d'animation de la rue (utilisant des vidéos en boucle) ainsi que les ambiances sonores, auraient également dû être automatiquement édités mais n'ont pas été pris en compte par manque de temps.

L'édition d'un espace urbain regroupe différentes étapes que nous avons considérées et pour lesquelles nous avons développé des outils. Chacune de ces étapes (schématisées dans la figure 4.1), doit fournir à la fin du processus une scène complète. Cependant, le module permettant d'associer les différents éléments (c'est-à-dire le module intégrateur) n'a jamais été terminé. L'intégrateur fédère les autres modules. Puisqu'il n'a pas été complètement développé, tous les modules ne peuvent fonctionner que de manière autonome. Ce n'est cependant pas tout à fait vrai puisque le module bâti utilise le module multimédia pour la sélection des textures à appliquer aux différents immeubles. Intégrer tous les modules dans une application fédératrice les utilisant comme différents *plugins* n'est pas la plus grande partie du travail. La partie la plus délicate concerne plutôt la récupération des deux scènes (du terrain et du bâti) pour former la scène finale.

Avec ce simulateur urbain, nous nous sommes aperçus que la navigation dans une scène 3D, décrite en langage VRML et visualisée avec un *plugin* associé, était acceptable en terme de fluidité dès lors que la scène 3D comportait un nombre raisonnable de polygones et de textures. Dans le cadre du D.E.A., nous avons voulu élargir ce principe à la visualisation d'un fonds ancien en pensant que l'espace 3D permettait d'organiser les ouvrages pour, d'une part, réduire les mouvements de l'utilisateur et, d'autre part, classer les livres selon certains critères de recherche. S'agissant de livres anciens dont le dos est une information essentielle pour aider les spécialistes dans leurs recherches, nous devions modéliser les livres en autant de faces texturées, ce qui causait un effondrement rapide des performances avec l'augmentation du nombre de livres [Cubaud98]. Bien que dans ce contexte particulier la puissance requise pour l'utilisation d'une interface 3D n'ait pas été suffisante et contrariait ce projet, l'aspect d'organisation des informations dans l'espace nous a séduit d'où la poursuite de ce travail par la proposition de ma thèse portant plus généralement sur les interactions 3D dans les systèmes d'informations réparties.

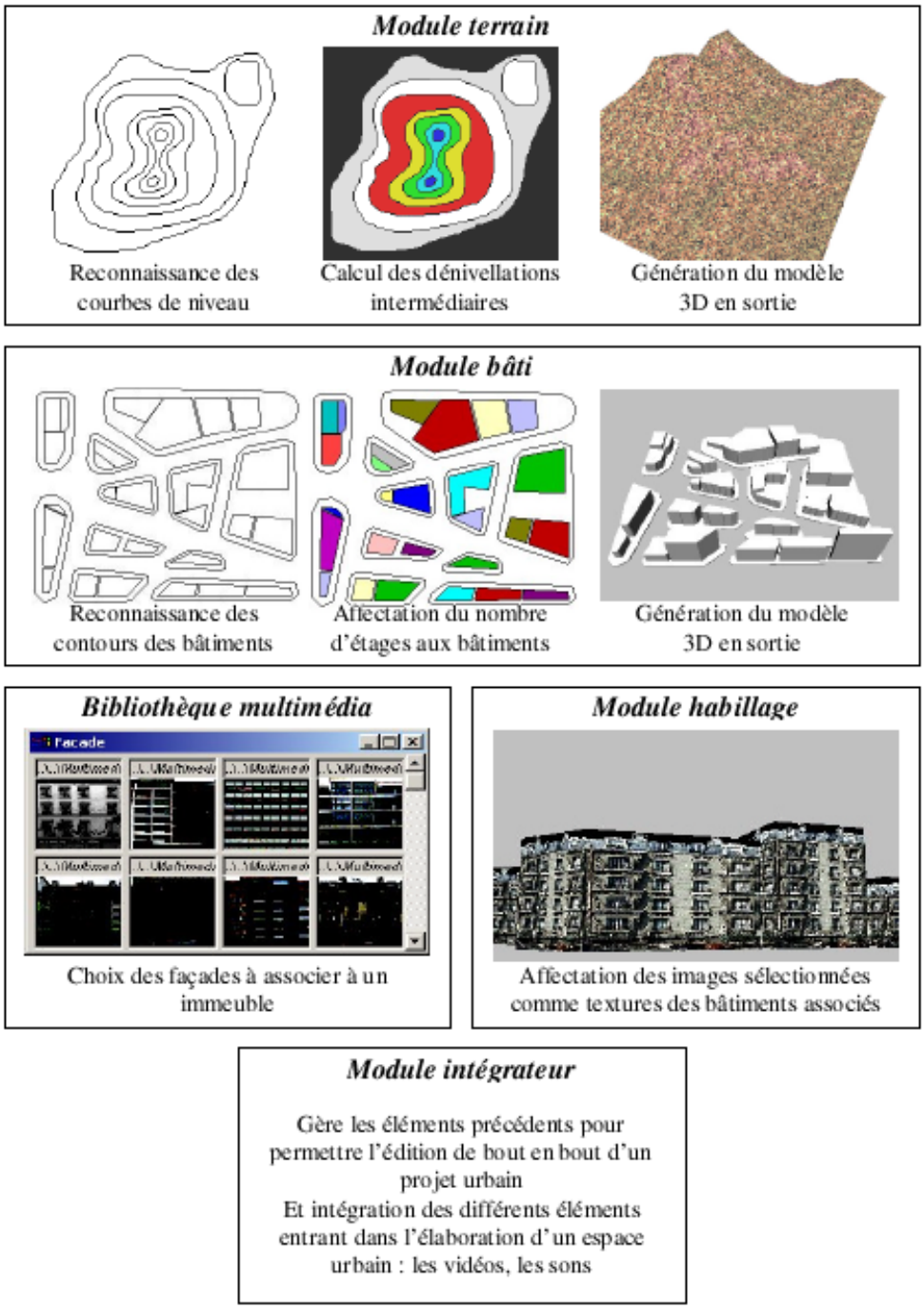


**Fig. 4.1** Les modules du simuateur urbain.

Alors que la thèse vise plus particulièrement à effectuer une étude théorique des interfaces 3D et de leurs applications, le mémoire d'ingénieur CNAM [Topol01], a permis d'étudier plus spécifiquement les outils de la 3D

## 4.3 Les plateformes d'expérimentation

Pour expérimenter les différentes applications, il nous suffisait en principe d'utiliser un navigateur VRML standard. La plupart de nos scènes et de nos scripts (lorsqu'ils sont entièrement compatibles avec la norme VRML) ont d'ailleurs été testés avec le navigateur Cortona de la société ParallelGraphics. Cependant, le problème posé par cette solution est l'impossibilité, d'une part, de faire des mesures de performance des scènes produites et, d'autre part, de pouvoir étendre ou modifier le langage VRML en fonction des besoins. Dans le cas où nous utilisons un produit commercial, il est en effet obligatoire d'accepter la manière dont sont implémentés les traitements des différents nœuds du langage VRML. Pour ces raisons, nous avons développé deux navigateurs "maison" avec lesquels il nous était possible d'inclure toutes sortes d'extensions ou d'intégrer des optimisations. Le premier navigateur, écrit en langage C et s'appuyant sur les deux API 3D OpenGL et Direct3D[1], et le second, écrit en Java et utilisant pour le rendu 3D soit Java3D soit un *binding*[2] OpenGL, ont tout deux fait l'objet de mémoires d'ingénieur au CNAM [Abisror02, Topol01].

Les deux projets de navigateur VRML ont été développés sur une plateforme Linux installée à partir de la distribution SuSE. Cette distribution, particulièrement robuste et stable, a très tôt, intégré la prise en charge des cartes accélératrices 3D. Il est amusant de noter que le succès de la 3D sous Linux ressemble étrangement à celui de Linux lui-même. En effet, la principale avancée est le travail d'un seul homme, Brian Paul, qui a mis à disposition sa librairie Mesa, utilisant une syntaxe et un automate semblables à ceux d'OpenGL (avec l'autorisation de Silicon Graphics). Créé en 1993, ce clone d'OpenGL a également connu un intérêt croissant qui a mené à de nombreux ajouts et de nombreux portages sur différentes plates-formes. Maintenant que l'accélération graphique est également présente, Linux est devenu une plate-forme idéale pour le développement d'applications 3D.

L'accélération des fonctions 3D sous Linux revêt deux formes. La première, et la plus ancienne, utilise une interface permettant d'effectuer un rendu accéléré d'une application 3D utilisant Mesa dans une fenêtre X11. Cette interface, appelée *Accelerated GLX*, est une mise à jour de GLX (*Graphic Library eXtension*) pour accéder aux fonctions 3D câblées sur certaines cartes graphiques. Ce portage a été possible grâce à Silicon Graphics, encore une fois, qui a rendu publiques les sources de GLX. Il suffit, pour en profiter, de trouver un driver GLX pour la carte graphique installée.

La deuxième forme d'accélération 3D sous Linux nécessite l'utilisation de la DRI (*Direct Rendering Infrastucture*) de Precision Insight. Cette solution est en fait un complément de la première. Elle intègre, en effet, le module GLX (accéléré ou non ; cela dépend de la carte graphique) directement dans le serveur X mais offre, en plus, un chemin optimisé pour envoyer les ordres de rendu 3D directement aux cartes graphiques supportées. Le serveur XFree86 4.0 intègre cette architecture de rendu direct qui est schématisée dans la figure 4.2. Bien évidemment, cette architecture ne pourra être utilisée que pour un rendu

[1] L'implémentation du navigateur avec les fonctions Direct3D a uniquement été faite pour pouvoir accéder à une accélération graphique efficace sur les plateformes Win32.

[2] Un *binding* est une liaison pour permettre à un langage d'accéder à des fonctionnalités d'un système d'exploitation. Les *bingings* OpenGL en Java, par exemple, autorisent l'utilisation de fonctions OpenGL à travers la JNI (*Java Native Interface*).

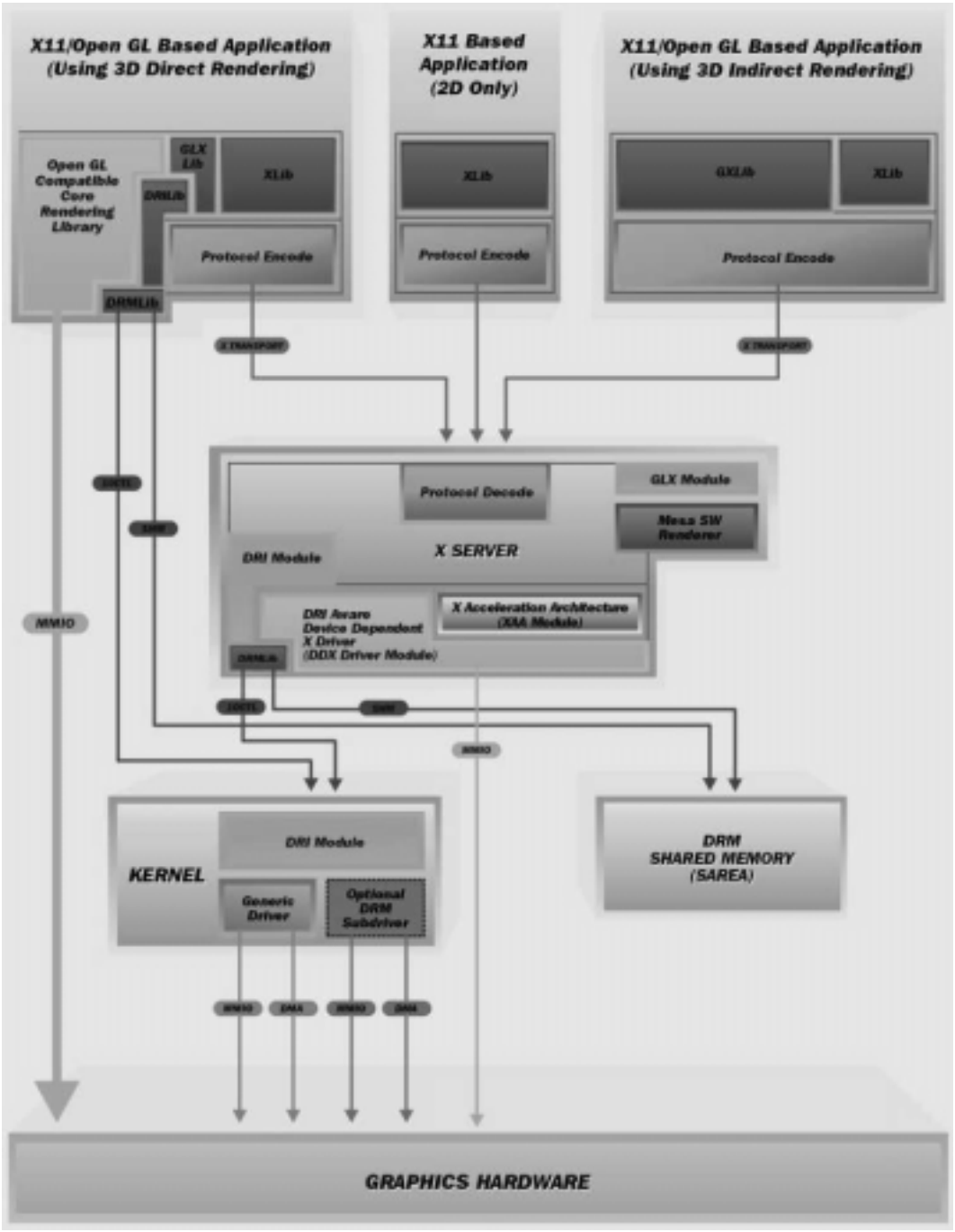


**Fig. 4.2** L'architecture DRI de XFree86 4.0.

local (le client et le serveur OpenGL étant sur la même machine). Pour le rendu sur une machine distante, la carte graphique de la machine effectuant les calculs, ne peut être utilisée. Cette architecture propose à peu de choses près, l'équivalent des couches HAL et HEL sous Windows (voir la présentation de Direct3D page 98) pour la prise en charge

des fonctions 3D. C'est-à-dire que lorsque la carte prend en charge une fonction, elle est accélérée ; sinon, elle est émulée par le microprocesseur.

Cette nouvelle architecture souffre encore de quelques petits problèmes. Pour la plupart, ils proviennent de sa relative jeunesse et du fait que son développement soit *Open Source*. Les *drivers* et applications libres de droits requièrent généralement un léger temps d'adaptation pour parvenir à leur maturité. Actuellement, XFree86 4.0 n'est fourni dans les distributions qu'à titre d'environnement expérimental. La version 3.3.6 de XFree86 fait office de serveur X par défaut. Les problèmes de XFree86 4.0 sont cependant répertoriés et en voie d'être réglés dans sa prochaine version. A l'heure où est écrit ce rapport, on note les problèmes suivants :
- le setup utilisant une interface graphique n'est pas implémenté,
- la configuration est encore difficile, s'adressant aux utilisateurs experts,
- l'accélération ne fonctionne pas encore avec toutes les cartes graphiques,
- le rendu direct ne fonctionne pas encore avec toutes les applications,
- les drivers DRI sont plus lents que les drivers GLX de la version 3.3.6 dans certains cas et pour certaines cartes graphiques.

### 4.3.1 Le langage VRML et son interpétation

Comme nous le disions précédemment, notre langage cible a été VRML. Le choix de ce langage a été motivé par différentes raisons dont les principales sont la disponibilité d'une spécification claire [VRML97] et l'existence de nombreux *plugins* (contrairement à MPEG4). Par ailleurs, et c'est la cause de la première raison, le langage VRML est la seule norme publique de langage de description de scènes 3D interactives.

La première étape du processus menant à la visualisation d'une scène décrite dans un fichier au format VRML, est la même quels que soient les applications ou les *plugins* écrits. Il s'agit de lire le fichier source, de déterminer si tous les mots utilisés font partis de la syntaxe VRML et si l'ensemble de ceux-ci forme une expression valide. Ce travail est pris en charge par le sous-système "Chargeur de scènes" (voir la partie 3.3.2 page 93). Plus généralement, cette première étape est une constante lorsque l'on développe un traducteur ou un compilateur.

Connaissant les différentes unités lexicales et la grammaire des expressions bien formées d'un langage, il est relativement facile de mettre en œuvre un traducteur. Dans la plupart des cas (et dans le nôtre aussi), les expressions bien formées sont traduites en une structure arborescente interne (en mémoire vive, par opposition au fichier source qui est stocké sur un disque) exploitable et manipulable plus facilement.

Nous avons donc créé deux interpréteurs (*parsers*) de fichiers VRML. Le premier a été réalisé en traduisant la grammaire décrite dans la spécification de VRML97 [VRML97] en une grammaire conforme à l'outil Yacc, qui associé à Lex, permet de mettre en œuvre très facilement des traducteurs et compilateurs. Pour ce premier *browser*, le processus menant du fichier VRML jusqu'à son affichage, est schématisé dans la figure 4.3. Dans le deuxième navigateur, le *parser* a été écrit en utilisant le compilateur de compilateurs JavaCC (*Java Compiler Compiler*). Cet outil écrit en Java effectue le même travail que Yacc et Lex.

L'avantage majeur d'un tel outil est que les grammaires VRML, VRMLScript ou encore X3D, ont déjà été écrites, rendant ainsi le développement de l'interpréteur VRML plus facile.

### 4.3.2 Le rendu graphique par OpenGL

La deuxième étape consiste à traduire la structure interne issue de l'interprétation du ou des fichiers VRML en la représentation visuelle correspondante. Pour cela, nous avons mis en place une fonction récursive parcourant l'arbre de nœuds créé en mémoire. A chacun des nœuds parcourus, il suffit alors de faire correspondre une série de commandes OpenGL (ou OpenAL pour les nœuds sonores). Les nœuds capteurs ou interpolateurs, responsables des interactions et des animations, ne génèrent pas directement des commandes OpenGL. Cependant, un traitement spécial est requis pour les garder en mémoire et y accéder rapidement pour tout déclenchement d'animation. Les structures utilisées (schématisées dans la figure 4.4) sont relativement complexes mais permettent d'accéder rapidement aux capteurs et horloges, et de répercuter efficacement leurs changements d'états.

A cause des animations éventuellement présentes dans un fichier VRML, il est impossible d'utiliser le mode retenu d'une API 3D pour en effectuer l'affichage. Comme nous l'avons expliqué dans les sections traitant des API 3D (voir page 94), les commandes graphiques du mode retenu ne peuvent que difficilement être mises à jour. Nous avons donc utiliser le mode immédiat, ce qui implique de parcourir l'arbre des nœuds VRML, autant de fois que l'on trace la scène. Ces nombreux parcours sont coûteux mais permettent, néanmoins, de visualiser des scènes interactives.

En C, il n'y a aucun problème particulier pour accéder aux fonctions OpenGL. En Java, les paquetages permettant d'effectuer le rendu, sont les *OpenGL bindings*. Ceux-ci ont pour but de lier une application Java avec les fonctions OpenGL natives rassemblées sous forme d'une bibliothèque dynamique. Il existe de nombreuses implémentations de ces *bindings*. La plupart sont disponibles sur différentes plates-formes. L'ensemble grammaire de description en JavaCC et *bindings* OpenGL est une alternative à Java3D. Ses principaux avantages viennent des nombreux systèmes supportés, ne les limitant pas à ceux pour lesquels une implémentation de Java3D existe. De plus, il permet un plus grand contrôle sur la génération des commandes 3D. Enfin, cette solution préférant OpenGL à Java3D nous permet d'éviter les problèmes que connaît Java3D pour gérer la mémoire allouée aux textures. Nous avons effectué des tests simples en 2D et en 3D, illustrés dans la figure 4.5, pour mettre en évidence ce problème. En 2D, le ramasse-miettes de Java fonctionne très bien mais en 3D, la mémoire allouée n'est jamais libérée. On remarque pour Java 2D que la mémoire allouée fluctue en fonction du nombre d'images. Au contraire, avec Java3D, la mémoire utilisée croît toujours même après libération des images non nécessaires.

### 4.3.3 Spécificités de nos navigateurs VRML

Bien que délicates et d'extrême importance, les étapes de lecture du (des) fichier(s) VRML composant une scène et leur traduction en fonctions 3D d'une API, ne forment qu'une première étape laborieuse mais sans réelles difficultés du travail. La seconde, concerne la

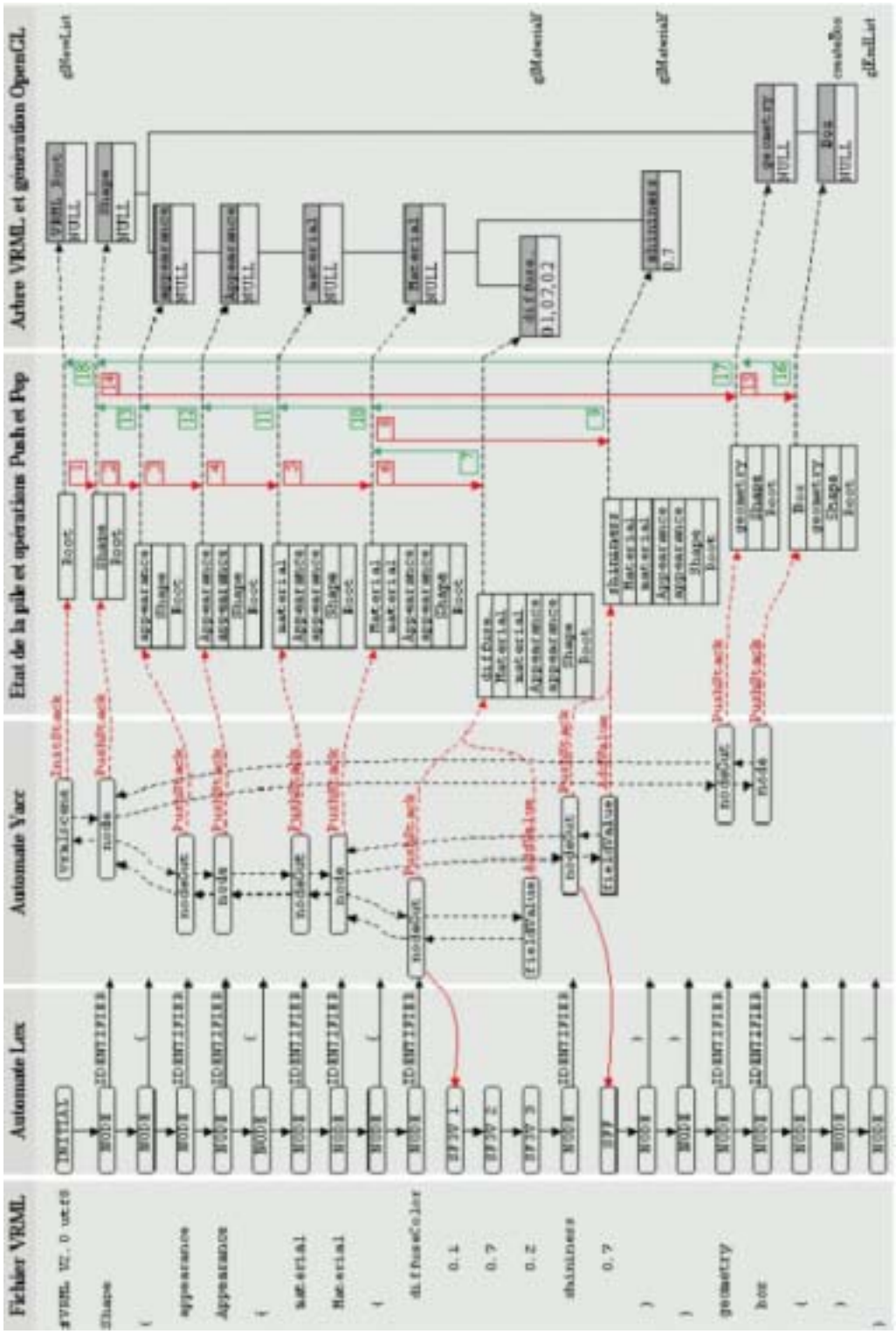


**Fig. 4.3** Le fonctionnement de notre interpréteur Yacc/Lex.

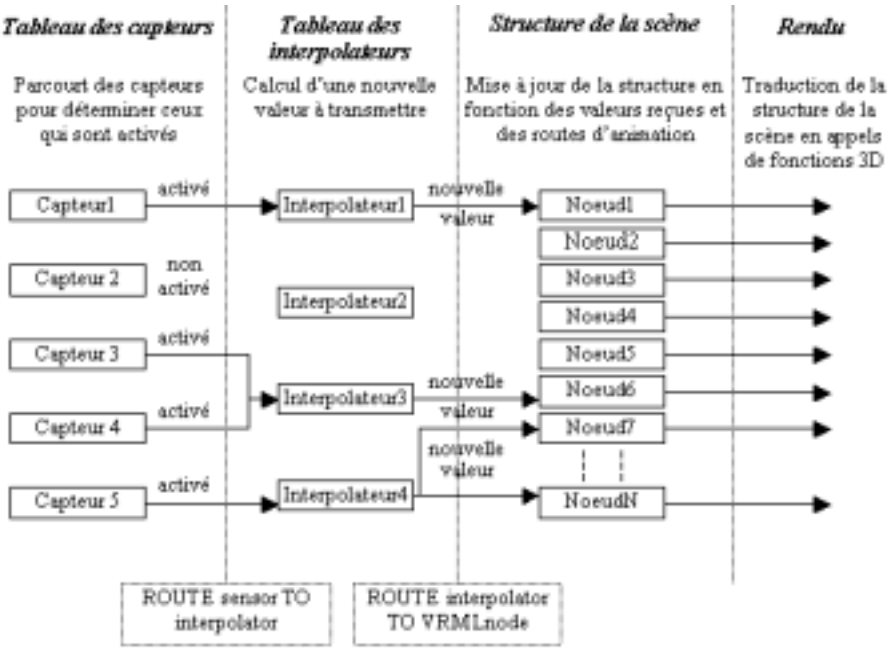


**Fig. 4.4** La gestion des capteurs dans notre navigateur VRML.

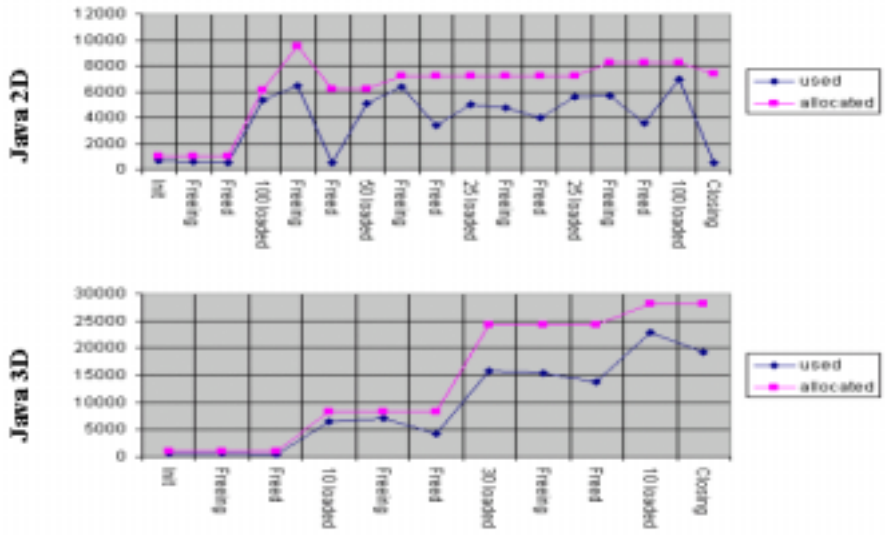


**Fig. 4.5** La gestion de la mémoire allouée pour les textures avec Java.

programmation d'outils pour permettre à un utilisateur d'évoluer dans la scène et d'en contrôler le rendu. Comme la plupart des clients VRML que l'on peut trouver sur Internet, nos navigateurs intègrent des options pour déplacer et orienter le point de vue sur la scène ainsi que pour changer le mode de tracé des objets.

Chacun de nos navigateurs VRML avait pour but de remplir des services particuliers. Le premier, écrit en C, devait permettre un affichage rapide des scènes statiques et proposer des outils de navigations avancés. Le deuxième, écrit en Java, visait avant tout à autoriser le rendu de scènes interactives et fortement scriptées. Il y a eu néanmoins un certain chevauchement puisque les deux sont capables, à présent, de gérer les scènes animées. Nous décrivons, par la suite, les principaux outils de navigation offerts dans le premier de nos navigateurs.

Le *browser* VRML écrit en Java et permettant le rendu soit à travers Java3D soit par l'intermédiaire d'OpenGL, dispose d'un avantage particulier. C'est le seul à pouvoir gérer les scripts écrits en Java. En effet, son architecture logicielle respecte parfaitement la hiérarchie des classes nécessaires pour exécuter les scripts Java dans une scène VRML. Cette fonctionnalité a été possible, avant tout, parce que Java est un langage objet et qu'il permet donc de créer cette même hiérarchie de classes. Notre premier navigateur, écrit en C, ne dispose pas de cette organisation objet. Pour y intégrer les scripts Java, il faudrait lui adjoindre une interface écrite en Java, qui serait grandement facilitée si nous avions utilisé le langage C++.

## Manipulation de la caméra

Pour les opérations de navigation, nous avons opté pour des méthodes plus originales tirées de travaux récents. Nous les avons préférées au paradigme de navigation exploitant des *widgets* de visualisation (voir page 85) utilisé dans les autres clients VRML. La navigation s'effectue à la souris, bien que les translations soient possibles à partir de certaines touches du clavier. La souris n'offrant que deux degrés de liberté, nous avons implémenté la technique présentée par MM. Zeleznik et Forsberg [Zeleznik99] permettant de contrôler une caméra virtuelle 3D sur les 6 degrés de liberté. Cette technique est présentée dans la section 3.4.1 page 121. Dans l'interface de notre navigateur, il n'y a donc aucun bouton pour effectuer la navigation. La surface d'affichage réservée au rendu de la scène dispose alors de la taille maximale.

UniCam, la technique de navigation à la souris que nous avons choisie, intègre huit techniques de manipulation de caméra regroupées dans 3 types de navigations différentes. Le choix de la correspondance entre les mouvements de la souris et ceux du point de vue est déterminée uniquement par un clic sur un bouton de la souris et par le début du mouvement initié par l'utilisateur. Les gestes, permettant de choisir ces différentes techniques, ont été choisis empiriquement, d'après les réactions des utilisateurs. Ces huit techniques (en italique), rassemblées dans trois familles d'interactions, sont les suivantes :

- Contrôles de la caméra à l'initiative de l'utilisateur
  - Translations de la caméra
    - *Translations X et Y*
    - *Translations X et Z*
  - Mouvements en orbite de la caméra
    - *Rotation autour du centre de la scène*
    - *Rotation autour d'un point quelconque*
- Mouvements automatiques

  - *Vue oblique sur la scène*
  - *Vue englobant la région sélectionnée*
- *Sauvegarde de la position de la caméra*
- *Restauration de la position de la caméra*

## Aides à la navigation

Dans UniCam, certains mouvements permettent de sauvegarder la vue courante (sous forme d'une image) et de la rappeler. Nous avons préféré aux images statiques utilisées dans UniCam, des WIMs (*Windows In Miniature*) présentées dans [Stoakley95]. L'utilisateur interagit dans les WIMs avec les mêmes mouvements de souris que dans la vue principale. Elles offrent ainsi de multiples vues sur la scène, ce qui apparaît très pratique lorsque la scène est conséquente. Dans la figure 4.6, deux WIMs ont été ouvertes, l'une en haut à droite (dont le cadre est apparent car la souris se trouvait à l'intérieur au moment de la capture) et l'autre en bas à droite. Chacune d'elles montre une vue différente de la vue principale.

**Fig. 4.6** Les WIMs [Stoakley95] de notre navigateur VRML.

Les WIMs que nous avons implémentées peuvent être redimensionnées, déplacées et supprimées. Ces interactions s'opèrent toujours à l'aide de la souris. Elles se rajoutent donc à celles permettant de manipuler la caméra avec les concepts d'UniCam. La différence entre ces deux types vient des calculs nécessaires. Pour les mouvements de caméra, les déplacements 2D de la souris sont transformés en translations ou en rotations 3D du point de vue. Pour la manipulation des WIMs, tous les calculs restent en 2D puisqu'elles ne sont pas inclinables. La gestion des différents états de la souris est donc un point critique et, particulièrement, la détermination des positions où ont lieu les clics.

En plus des WIMs standard, nous avons ajouté la possibilité de définir des loupes sur la scène (voir figure 4.7). Ces loupes ont la même apparence que les WIMs, c'est-à-dire qu'elles sont incluses dans des conteneurs 2D manipulables. Elles sont particulièrement pratiques pour obtenir les détails sur les objets, sans devoir déplacer le point de vue.

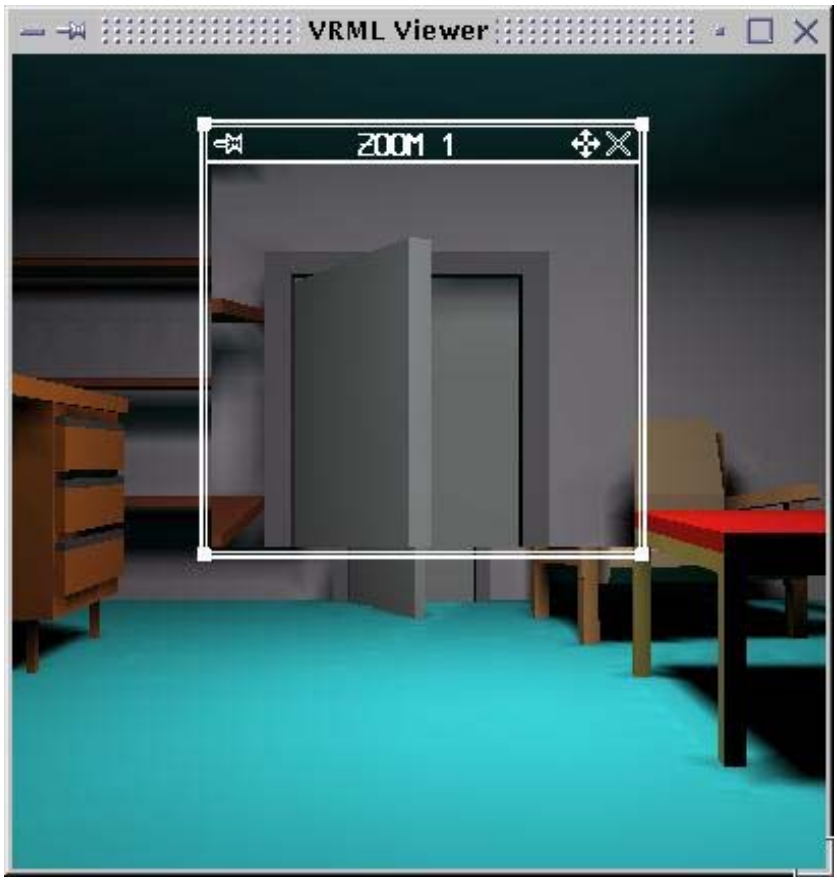


**Fig. 4.7** Exemple d'aides à la navigation : les zooms.

## 4.4 Bibliothèques numériques

Maintenant que l'on sait sauvegarder tout livre, toute musique, tout film, ..., sous un format numérique et qu'il est possible de les consulter via Internet, le vieux rêve de H. G. Wells[3], datant de 1938, d'avoir une bibliothèque comportant toutes les cultures présentes et passées du monde, semble presque une réalité. Une telle bibliothèque serait labyrinthique et ressemblerait fort à la bibliothèque de Babel imaginée par J. L. Borges[4]. Cette bibliothèque porte le même nom que la Tour à l'origine selon la légende, de la séparation des hommes par des langues différentes. Elle serait le rassemblement de toutes les cultures, rompant ainsi la punition divine infligée à l'humanité. Le numérique donne la possibilité de recréer la bibliothèque d'Alexandrie, au XXème siècle, tout en faisant s'écrouler les murs de la bibliothèque de Babel selon notre propre volonté, en proposant des éléments multimédias répartis sur Internet.

Jusqu'à une certaine taille de contenu, construire de telles bibliothèques n'est pas un problème. Le livre de Mickael Lesk [Lesk97] explique d'ailleurs, comment les mettre en œuvre. L'avantage majeur des bibliothèques numériques est la possibilité de consulter des ouvrages depuis son ordinateur. Ceci paraît plus que nécessaire lorsque la bibliothèque contenant les livres que l'on veut consulter, se trouve à l'autre bout du monde. Cette possibilité de consulter les ouvrages à distance est particulièrement utile dans les cas des fonds anciens. En effet, il est plutôt difficile, voire impossible certaines fois, d'obtenir le droit de consulter ces fonds. Cette protection est légitime puisqu'il peut s'agir de livres uniques sur lesquels les dégâts subis seraient irréparables. Une bibliothèque numérique autorise tout le monde, et non plus seulement une élite, à consulter ces ouvrages à distance, sans le moindre risque de les abîmer.

Une bibliothèque numérique a la possibilité d'être réorganisée facilement. Les classements et les recherches sont en effet très rapides. Il est, par exemple, possible de reclasser en quelques secondes tous les livres de la bibliothèque par date de parution au lieu du nom de l'auteur. Avec une bibliothèque traditionnelle, ce reclassement durerait des jours. On peut également dupliquer les livres pour les classer selon plusieurs critères différents. Cette duplication numérique n'engendre pas de coûts supplémentaires alors que, dans une bibliothèque réelle, il faut autant d'exemplaires du livre que de critères de classement.

Une bibliothèque peut être décrite par les quatre activités suivantes, chacune d'elles étant automatisée si la bibliothèque est numérique :
- Collection : rassemblement des ressources en fonction des informations auxquelles elles répondent.
- Organisation et représentation : classement et indexation des informations pour aider les utilisateurs.
- Accès : organisation physique de l'espace.
- Analyse, synthèse et propagation de l'information : réponse à des questions de référence, production de dossiers d'évaluation.

[3] Auteur britannique de science-fiction du XXème siècle.
[4] J. L. Borges, Fictions, Collection Folio, Gallimard, 1983.

### 4.4.1 Contexte de cette étude

Pour expérimenter l'utilisation de la 3D et son apport pour la recherche d'ouvrages dans un corpus et la lecture des livres, nous disposons, au Conservatoire National des Arts et Métiers, de deux bibliothèques numériques qui sont opérationnelles. Chacune d'elle comporte ses spécificités aussi bien sur le type d'ouvrages qu'elles proposent, que sur leur format de numérisation. Ces deux caractéristiques sont essentielles pour déterminer l'utilité d'une interface spatialisant les données.

Ces deux bibliothèques, l'ABU (Association des Bibliophiles Universels) et le CNUM (Conservatoire NUMérique des Arts et Métiers), nous ont permis d'effectuer différentes expérimentations sur les interfaces de recherche et de lecture. Par ailleurs, elles nous fournissent des informations (sous forme de *logs*) intéressantes pour déterminer les usages qu'en font les utilisateurs des livres auxquels ils peuvent accéder. Nous avons pu remarquer, par exemple, que sur ABU la majorité des lecteurs préfèrent télécharger complètement un livre plutôt que de le lire en ligne. Ainsi, par ces rapatriements des livres sur leur poste de travail, ils peuvent les transformer en un format qui leur convient mieux pour lire que le format texte brut. Certains doivent même en profiter pour imprimer le livre.

Pour le CNUM, nous n'avons pas permis le téléchargement d'ouvrages. Puisque nous présentons des fac-similés (chaque page est un fichier GIF d'une centaine de kilo-octets), il serait en effet bien trop long de télécharger un livre complet. Nous avons pu constater qu'en moyenne un utilisateur ne lisait que quelques pages (deux ou trois) puis qu'il quittait le site. Dans les deux cas, cela nous conforte dans l'idée qu'il faut fournir à l'utilisateur un moyen de lire plus efficace, plus ludique, plus accrocheur.

#### L'ABU

ABU, la première bibliothèque mise en place au CNAM est constituée de livres numérisés en mode texte. Ces ouvrages mis en ligne ont été saisis ou numérisés par des volontaires et sont donc, pour la plupart, les romans préférés des pupitreurs. ABU ne peut donc pas suivre une politique éditoriale déterminée puisque cette bibliothèque dépend des goûts et des soumissions de ses bienfaiteurs. Et l'on ne peut, en aucun cas, demander à un volontaire de saisir un texte qui ne l'intéresse pas. Il est déjà bien assez pénible de saisir un texte que l'on apprécie !

La mise à disposition des livres au format texte, apporte un certain nombre d'avantages. On peut noter, en particulier, la possibilité de modifier les textes pour les présenter par la suite sous différents formats et sur différents supports. Il est possible, par exemple, d'utiliser les textes d'ABU pour nourrir un logiciel de synthèse vocale et les rendre ainsi accessibles aux mal-voyants. Ils peuvent également être transformés en langage braille, ce qui est impossible avec les documents numérisés sous forme de fac-similés (c'est-à-dire en mode images). Toutefois, la plupart des utilisateurs ont l'habitude de lire les textes sous un format plus accessible qu'ils préfèrent. C'est pourquoi, dans le cadre d'un mémoire d'ingénieur CNAM, nous sommes en train d'intégrer l'exportation des textes au format PDF (*Portable Document Format*).

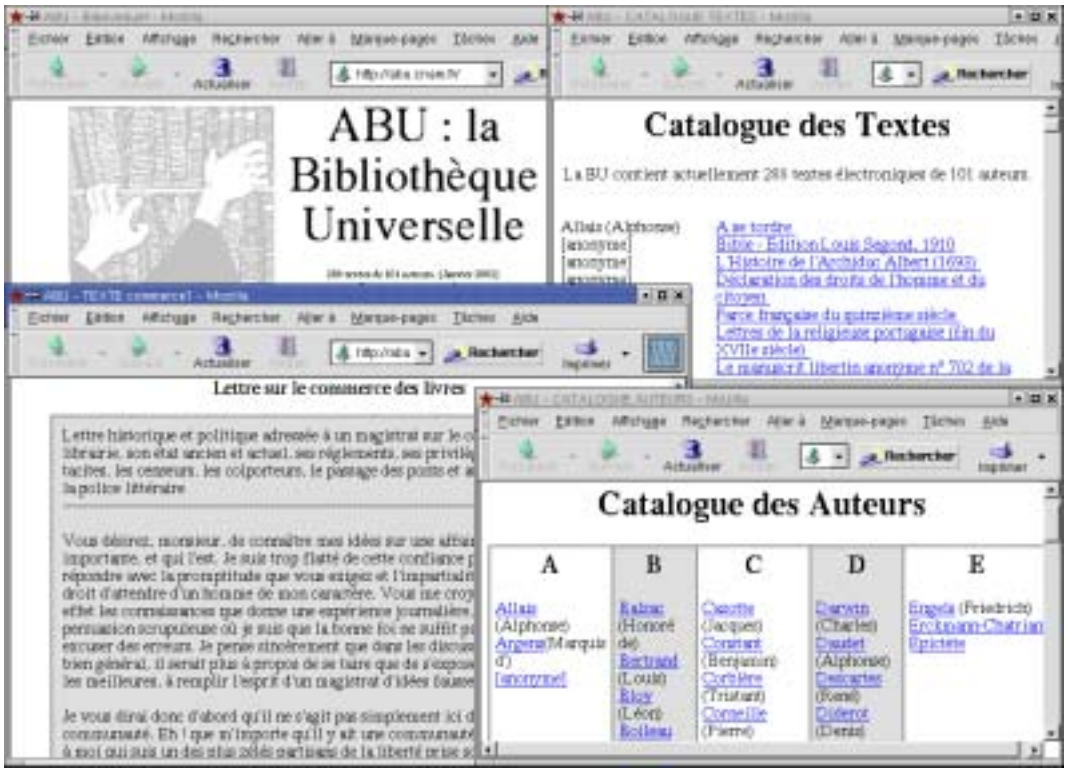


**Fig. 4.8** Une copie d'écran de la bibliothèque numérique ABU.

ABU ne fournit donc pas d'informations sous forme d'images et ceci ne favorise pas la création d'une interface graphique. Certes, les informations disponibles au format texte apportent des avantages pour effectuer des recherches d'occurrences, dans le livre lui-même. Cependant, les textes sont difficilement "spatialisables" car le rendu de texte en 3D est coûteux. Bien que nous n'ayons aucun intérêt à créer une interface 3D pour la lecture des textes, nous avons pu néanmoins expérimenter une interface pour spatialiser les résultats des requêtes. A la manière des interfaces gérant les proximités sémantiques par mesures de similitudes entre documents, les livres sont répartis en fonction de la manière dont ils répondent aux requêtes des utilisateurs.

Mais, la lecture s'effectue quoi qu'il en soit, toujours "à plat", dans une interface classique. Or, nous ne pensons pas que le mélange d'interfaces (3D pour la recherche et 2D pour la lecture) soit une bonne idée. Il suppose le changement constant des métaphores de navigation, ce qui introduit une gêne pour l'utilisateur. Alors que le CNUM n'existait pas encore, ABU nous a fourni, dès 1998, le premier catalogue qui nous a servi pour notre première expérimentation sur l'interface de recherche.

Après avoir étudié les transformations des textes bruts en d'autres formats (dont PDF en particulier), il nous apparaît qu'il est possible de convertir les textes en images par les utilitaires Postscript et NetPBM (une suite d'outils pour la transformation d'images). Cette conversion nous ouvre de nouvelles perspectives intéressantes puisqu'il est possible à la fois d'accéder à la bibliothèque au moyen d'une interface 3D (en utilisant les images

produites comme des textures), mais aussi de conserver la recherche des occurrences dans le corpus entier, grâce au format texte (comme cela est schématisé dans la figure 4.9).

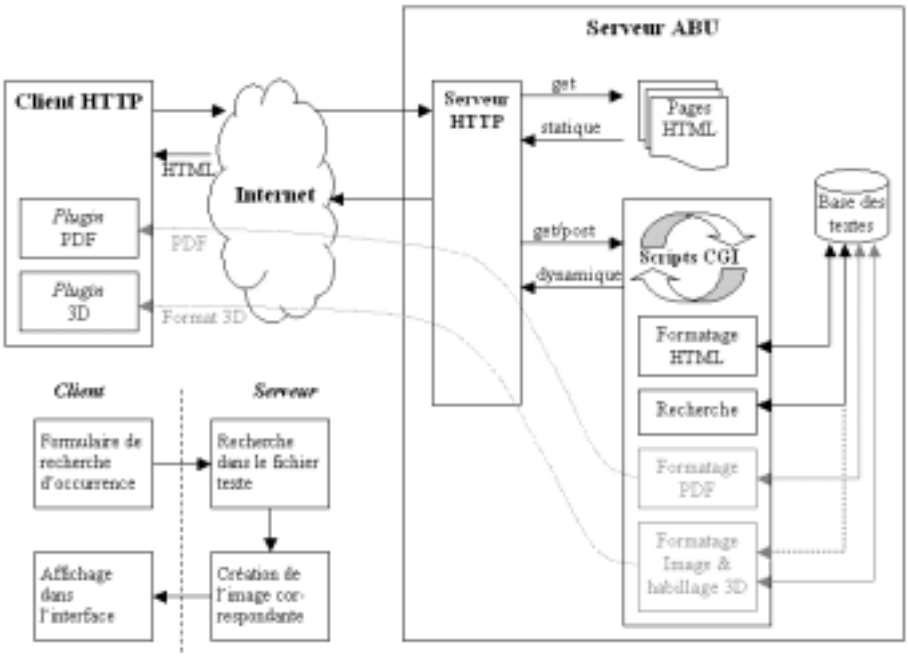


**Fig. 4.9** La recherche d'occurences envisageable avec l'ABU en mode graphique.

## Le CNUM

Le Conservatoire numérique des Arts et Métiers (CNUM), depuis février 2000, ouvre au public le fonds patrimonial de la Bibliothèque du Conservatoire National des Arts et Métiers (CNAM) : une collection d'ouvrages et de périodiques francophones significatifs, traitant des sciences et des techniques, de l'économie et de la sociologie appliquées, est consultable sur l'Internet[5]. Le Conservatoire Numérique s'adresse aux chercheurs et aux enseignants en histoire des sciences et des techniques, en épistémologie, en didactique, en offrant à la fois des textes et une documentation iconographique spécifiques. Le CNUM fait aussi acte de vulgarisation scientifique en remplissant l'une des missions fondamentales du CNAM : la diffusion du savoir et la reconnaissance du patrimoine scientifique et technique francophone.

La mise en œuvre de ce projet est le fruit d'un partenariat entre trois institutions du CNAM : la Bibliothèque, le Centre d'histoire des techniques (CDHT/EHESS) et le Centre d'études et de recherches en informatique (CEDRIC). Le CDHT et la Bibliothèque sélectionnent un ensemble cohérent et représentatif de livres et de périodiques relatifs à l'histoire des sciences et des techniques. La Bibliothèque assume ensuite la préparation scientifique et

[5]Accessible à l'URL `http ://cnum.cnam.fr`.

technique des documents. L'équipe "Multimédia et Interaction Homme-machine" du CEDRIC apporte son concours dans l'établissement, le stockage et la diffusion par Internet des documents numérisés. Elle est l'interlocutrice technique dans le choix des prestataires et assure la maintenance du site. L'équipe s'entoure, pour l'élaboration de chaque thématique, de la collaboration de chercheurs rattachés au CNAM ou à d'autres institutions.

Les documents numérisés sous forme d'images sont accompagnés d'outils bibliographiques permettant d'enrichir la consultation sur le site et d'ouvrir la recherche vers les autres textes ou les autres sites existants : bibliographies, catalogues analytiques, identification et localisation des autres documents consultables sur le sujet, avec le cas échéant, des liens hypertextes, etc.

Cette seconde bibliothèque numérique dont nous disposons se différencie de la première par le type de numérisation. Alors qu'ABU utilise uniquement des textes, le CNUM propose des fac-similés. Ces documents au format GIF sont faciles à utiliser dans une interface graphique, et, en particulier, dans une interface 3D dans laquelle ils peuvent faire office de textures. Cependant, contrairement aux textes d'ABU, les fac-similés ne permettent pas de faire des recherches d'occurrences sans adjonction d'un système de reconnaissance de caractères.

### 4.4.2 Les interfaces usuelles

La proposition d'une interface d'accès à une bibliothèque numérique dépend de certaines caractéristiques. Plus généralement, la conception d'une interface dépend de certains paramètres tels que le type des données, leur nombre, leur taille, leur structuration ou encore leur représentation. Nous avons vu dans le chapitre 2 qu'en effet les visualisations diffèrent en fonction de certaines de ces caractéristiques. Pour les bibliothèques numériques, nous devons prendre en compte non seulement la structuration séquentielle des livres, mais également la manière dont ils sont été numérisés (au format texte ou graphique).

Quelle que soit la bibliothèque numérique, une interface faisant le lien avec l'utilisateur est nécessaire. L'interface la plus simple est celle qui présente les documents dans une liste textuelle de liens (comme pour la bibliothèque ABU dans la figure 4.8). Mais l'intérêt des bibliothèques numériques, à partir du moment où elles dépassent une certaine taille, est de permettre à l'utilisateur d'interroger les catalogues à l'aide de requêtes qui peuvent être complexes. La mise en place de ce système de requêtes nécessite une interface pour la construction des requêtes et une interface de sélection des résultats. L'utilisateur peut alors sélectionner les livres qui l'intéressent afin de les consulter à l'aide de l'interface de lecture.

La manière usuelle de mettre à disposition de l'utilisateur ces différents outils, est d'écrire un certain nombre de scripts CGI (*Common Gateway Interface*) qui génèrent les listes d'ouvrages recherchés et l'interface de feuilletage. On voit déjà que les scripts sont omniprésents dans une telle interface ; et que cela n'est pas propre à la 3D. Les scripts CGI construisent des pages HTML présentant les informations demandées. On aura ainsi besoin de construire à la volée des pages HTML pour le catalogue et pour les pages de livres. Cette solution ne favorise aucunement la lecture croisée puisqu'une fenêtre est nécessaire

par page ouverte. L'interface se surcharge très rapidement comme on peut le voir dans la copie d'écran du CNUM dans la figure 4.16 page 159.

La 3D représente une alternative intéressante à étudier car les objets peuvent être spatialisés. Dans une telle interface, en profitant de la profondeur, de la transparence et de la rotation, il est possible de proposer un nombre nettement plus grand d'ouvrages. On peut considérer que les livres, le catalogue, les formulaires et les résultats de recherche cohabitent dans le même environnement 3D alors qu'en 2D il apparaît qu'ils sont en concurrence à cause de leur conteneur (la fenêtre recouvrante).

Par ailleurs, la manière usuelle de présenter les ouvrages disponibles dans une bibliothèque ne favorise pas la recherche visuelle. Les longues listes de livres classés selon leur titre ou leurs auteurs ne sont en effet pas idéales pour allécher le visiteur. Les interfaces graphiques permettent, grâce à leurs résolutions d'affichage et leurs nombreuses couleurs disponibles, d'étudier d'autres alternatives. Par exemple, la présentation des couvertures ou des dos de livres qui fournissent une information supplémentaire.

### 4.4.3 Les heuristiques visuelles

Notre but en utilisant les interfaces 3D est non seulement de permettre une spatialisation des données mais aussi de fournir aux utilisateurs des heuristiques de recherche visuelles. Il est particulièrement important de laisser l'utilisateur choisir des ouvrages en fonction de critères visuels que l'on peut considérer, de premier abord, comme étant secondaires. Or, de nombreux lecteurs se reposent justement, soit sur un jugement instinctif, soit sur leur mémoire photographique, pour trouver un livre d'intérêt. Ces deux méthodes, sont exploitables que si nous fournissons des informations visuelles.

La figure 4.10 illustre l'importance d'avoir une vision de l'objet livre plutôt que de disposer uniquement d'une description de ses caractéristiques. Un spécialiste des livres anciens peut tout à fait reconnaître certaines qualités des livres, comme leur taille, leur édition, leur format et par déduction, la date approximative de leur parution. Dans le cas de livres récents, comme ceux présents dans la figure 4.11, on utilise également notre vision et notre expérience pour identifier les livres sans même pouvoir lire leur titre. Par exemple, on distingue très bien que les petits livres forment une série de la collection "Que sais-je". Ou encore, qu'un autre livre est issu de la collection O'Reilly, bien connue des informaticiens.

En aucun cas, il n'est nécessaire d'adopter une interface 3D pour mettre en place des heuristiques de recherche visuelle. La copie d'écran de la figure 4.11 illustre ce propos. On y voit une page HTML standard dans laquelle sont inclues les dos des livres. Cette simple expérience nous montre à quel point il est facile alors de reconnaître certains ouvrages en fonction de leur taille ou du nombre de volumes. Cette présentation "virtuelle" des livres est tout aussi efficace que la vision directe des véritables livres de la collection Sartiaux. L'aspect le plus gênant dans cette interface est le zoom discret (par à-coup) mis en place.

**Fig. 4.10** Une partie de la donation de Sartiaux.

### 4.4.4 Les interfaces 3D

L'apport essentiel d'une interface 3D par rapport à l'interface précédente est la possibilité d'effectuer des zooms en temps-réel sur les livres. Par ailleurs, il est possible, avec la 3D, d'utiliser la profondeur pour répartir les ouvrages dans l'interface de recherche ou organiser son espace de travail dans le cas du poste de lecture. Par rapport à une interface zoomable telle que Pad++ [Bederson94], les interfaces 3D autorisent de surcroît des organisations utilisant les rotations, qui sont absentes des visualisations 2.5D.

Une bibliothèque numérique sera composée de trois blocs distincts. Le premier bloc intégrant l'organisation de la collection sera une interface permettant de sélectionner les documents, c'est-à-dire de les rechercher et de prendre connaissance de leur fiche signalétique. Le deuxième bloc, appelé poste de lecture, sera une interface pour lire les documents

**Fig. 4.11** Une interface zoomable en HTML.

sélectionnés. Le troisième bloc, quant à lui, autorisera l'utilisateur à annoter les ouvrages et à mettre à disposition des autres lecteurs ses propres remarques. Ce dernier aspect pourra être intégré au poste de lecture.

Les études que nous avons faites, présentent deux des trois interfaces nécessaires pour travailler sur un livre numérique en 3D. Ces interfaces, prototypées à l'aide du langage VRML, sont en trois dimensions afin d'exploiter au mieux l'espace de travail. Notre préoccupation principale, pour la construction de ces prototypes, était de s'affranchir des aspects qui nous limitent dans le monde réel. Notre but n'était donc pas de recréer une bibliothèque à l'identique mais de profiter de tout l'espace 3D disponible avec des contraintes minimales. C'est pourquoi, nous le verrons, dans notre premier prototype, les livres et l'utilisateur flottent au lieu de reposer sur des étagères ou sur le sol. Nous évitons également à l'utilisateur de reproduire des gestes qu'il effectuerait dans une véritable bibliothèque : se déplacer jusqu'aux ouvrages d'intérêt, les emporter jusqu'à une table et enfin les consulter. Les dé-

placements dans une scène 3D sont relativement difficiles pour les novices. Les limiter, à défaut de les rendre intuitifs, rend vraisemblablement service à ces utilisateurs.

## L'approche "réalité virtuelle"

Certaines bibliothèques "virtuellement réelle" comme [Kipp97] et [Plénacoste01], reproduisant à l'identique la bibliothèque réelle prise pour modèle, peuvent offrir d'appréciables services. Les habitués d'une telle bibliothèque retrouveront probablement sans problème les différents ouvrages, puisqu'ils auront l'impression (et c'est vrai) de connaître déjà les lieux. Cependant, un pourcentage infime des utilisateurs accédant à une bibliothèque en ligne sont dans ce cas. Pour la grande majorité, une représentation fidèle du lieu ne sera d'aucune utilité.

Si toutefois la bibliothèque est reconstituée en utilisant des livres texturés, l'utilisateur peut tout de même trouver des livres d'un simple coup d'œil. Dans le projet de Kipp *et al.* [Kipp97], les ouvrages disposés dans des salles, sur des étagères, sont habillés avec des couleurs unis distinctes signifiant qu'ils appartiennent à telle ou telle catégorie. Ce genre d'informations n'est absolument pas nécessaire et apparaît être bien moins utile que de fournir le dos du livre. Il est en effet impossible de discriminer un livre d'un autre dans une même catégorie. A la décharge de Kipp *et al.*, la bibliothèque qu'ils ont mise en ligne est une collection de thèses et d'articles dont le dos et la couverture ne donnent aucune information particulière.

En règle générale, la reproduction d'un univers réel tel qu'une bibliothèque avec ses salles, ses allées et ses rayons n'offre aucune amélioration notable et a souvent pour conséquence directe de nous focaliser sur les explorations déambulatoires, au détriment des recherches directes et spécifiques. Pour cette raison, nous avons décidé de prototyper une interface 3D aussi intuitive que possible, mais très éloignée de la représentation réelle des bibliothèques.

## L'approche "réalité augmentée"

Nombreuses sont les études proposant des méthodes – plus efficaces les unes que les autres – de navigation à l'intérieur d'un espace virtuel contenant un important nombre d'informations (voir chapitre 2). Ces différents travaux démontrent qu'un environnement 3D est efficace, avant tout, lorsque l'on profite des avantages offerts par la spatialisation. Pour cela, on peut et on doit s'affranchir de certaines contraintes du monde réel. Quelques aspects du monde réel peuvent toutefois être employés pour faciliter la compréhension de l'environnement virtuel. L'utilisation d'objets ou d'interactions issus du réel dans un environnement non réel est ce que nous appelons ici la réalité augmentée (définition qui diffère de la définition classique de l'expression "réalité augmentée")[6].

C'est ce mélange entre aspects réels bien maîtrisés et nouveaux services plus efficaces que nous avons tenté de mettre en œuvre. Dans l'interface de recherche dont nous parlons dans la section suivante, les livres présentés sont issus du fonds ancien du CNAM qui

[6] La réalité augmentée est généralement l'expression utilisée pour signifier que l'on rajoute des informations virtuelles au-dessus d'images réelles.

est difficilement accessible. Reproduire cet espace à l'identique n'a aucun intérêt. Il n'est d'ailleurs pas prévu pour être visité : les livres y sont stockés sur des étagères jusqu'au plafond. Nous préférons donc offrir des services autres, en espérant rendre les tâches de recherche et de consultation des ouvrages plus faciles.

### 4.4.5 Interface de recherche

Pour la sélection des ouvrages, la 3D permet non seulement d'éviter la solution à base de longues listes interminables, mais également d'offrir des informations visuelles supplémentaires grâce aux images des dos des livres. Elle offre également la possibilité de lier la navigation dans la bibliothèque avec l'élargissement ou l'affinement de critères de recherche. Ce que nous voulons mettre en place, c'est la création "à la volée" d'une nouvelle organisation des livres dans la bibliothèque, à chaque requête des clients. Les résultats seront alors séparés en groupes, ceux qui répondent à tous les critères, ceux qui répondent à tels critères et pas à tels autres, etc. Notre travail est alors d'offrir à l'utilisateur le moyen de passer d'un groupe d'ouvrages à un autre pour affiner ou élargir sa recherche.

Cette façon de construire et de soumettre les résultats à l'utilisateur, favorise la découverte visuelle puisqu'il est possible de regrouper les ouvrages ne répondant pas à tous les critères, à côté du groupe d'ouvrages pertinents. Or, cette manière d'aller "pêcher" un livre dans une bibliothèque est une heuristique très employée. On découvre souvent un livre intéressant par simple coup d'œil à côté de ceux que l'on recherchait vraiment. C'est pourquoi nous avons proposé cette présentation des résultats aux requêtes. L'apparence d'un livre (sa taille, son dos) devient alors une information aussi importante que son titre ou son auteur et nous facilitons l'appréhension de cette information par l'utilisation des textures des livres. De cette façon, nous autorisons le type de recherche décrite dans cette citation de A. Mengel :

> Les livres s'affirment grâce à leurs titres, leurs auteurs, leurs places dans un catalogue ou une bibliothèque, leurs illustrations de couvertures. [...] Je juge un livre à sa couverture ; je juge un livre à sa forme. [Mengel98]

La manière de retranscrire cette citation en une interface 3D a été présentée à la conférence ACM Digital Library 1998 [Cubaud98].

Pour créer une telle interface de recherche, nous utilisons un script CGI écrit en langage C. Il génère et renvoie dans une réponse HTTP, une scène VRML décrivant la bibliothèque organisée en fonctions des critères de recherche. Les critères de recherche sont saisis à l'aide d'un formulaire classique décrit en HTML, dont le bouton de soumission appelle le script CGI sur le serveur. Pour que ce formulaire de recherche soit consistant avec l'interface de recherche, il faudrait que nous imaginions une interface 3D pour la construction des requêtes.

#### Classement par groupes

Afin de clarifier l'idée de liaison entre la réponse à une requête sous forme de groupes et la navigation, considérons par exemple le résultat d'une recherche sur trois critères pour

trouver les ouvrages concernant les traités d'astronomie écrits au XVIII$^{eme}$ siècle et parlant de la planète Jupiter. La figure 4.12 montre, d'une part, les différents groupes d'ouvrages symbolisés par les nœuds et caractérisés par des opérations booléennes "et" et "non" sur les critères (vrai si les livres dans le groupe répondent au critère, faux sinon), et, d'autre part, les possibilités de navigation d'un nœud à un autre, symbolisées par les liens.

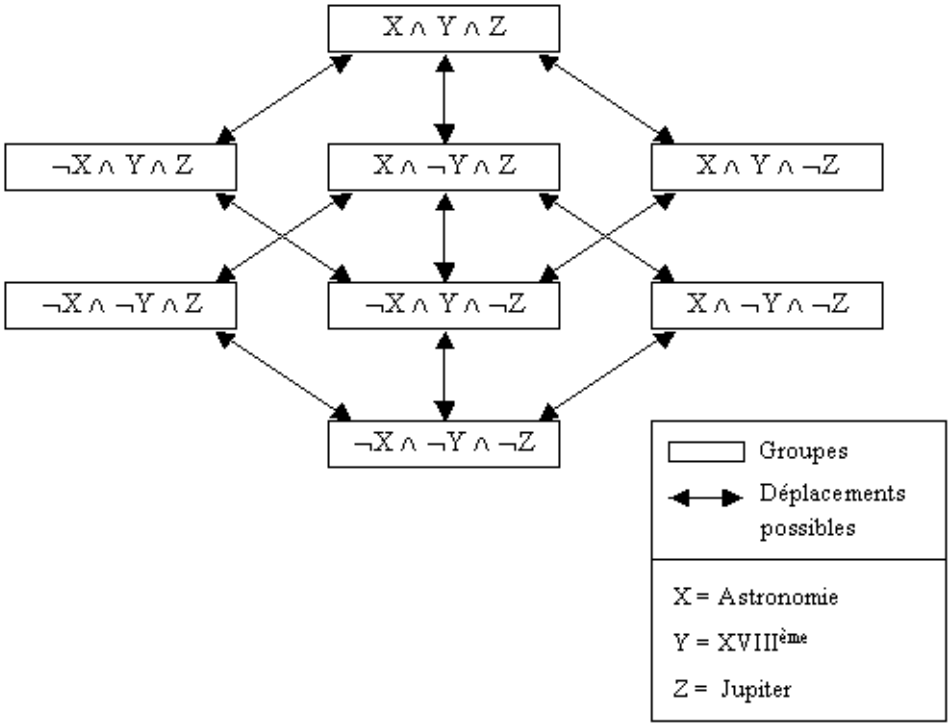


**Fig. 4.12** Classement et parcours possibles pour une recherche sur 3 critères.

Chacun des groupes est unique et leur nombre équivaut au nombre de permutations de l'ensemble des trois valeurs booléennes associées aux critères, de telle manière qu'aucun livre ne pourra figurer dans plusieurs groupes à la fois. On comprend, par exemple, que dans le groupe $X \wedge \neg Y \wedge Z$ se trouveront les livres d'Astrologie traitant de la planète Jupiter mais non écrits au XVIII$^{eme}$ siècle. Les déplacements possibles de l'utilisateur depuis cet exemple de groupe, pourront se faire selon 3 directions (la négation de chacun des critères), de telle manière qu'il pourra atteindre les groupes suivants :

- $\neg X \wedge \neg Y \wedge Z$ par négation du critère X,
- $X \wedge Y \wedge Z$ par négation du critère Y,
- $X \wedge \neg Y \wedge \neg Z$ par négation du critère Z.

De manière générale, il y aura pour $n$ critères de recherche :

- $2^n$ nœuds ou groupes de livres,
- $\sum_{p=0}^{n} C_n^p.p$ liens ou chemins de navigation.

De chacun des nœuds de ce graphe il y aura $n$ chemins possibles, correspondants à la négation d'un des $n$ critères.

## Spatialisation des groupes

Nous avons imaginé plusieurs représentations à partir de ce modèle. La première solution envisagée fut de représenter les groupes sous la forme d'un arbre, la racine étant le groupe des livres totalement pertinents. L'utilisateur aurait alors eu la possibilité de descendre ou de remonter le long d'un chemin pour affiner ou élargir sa recherche. Cependant, cette solution ne bénéficie pas des apports de la $3^{eme}$ dimension et oblige l'utilisateur à remonter des chemins au lieu d'emprunter tous ceux de la figure 4.12.

### *Dans un atomium ou un cylindre*

Nous avons alors pensé à un système presque équivalent mais dans une espèce d'atomium comme celui de la figure 4.14. Chacune des boules correspond à un groupe de livres. L'utilisateur est placé initialement dans la boule de l'atomium rassemblant les livres qui répondent à tous les critères ; il a alors la possibilité de se déplacer directement d'un groupe à l'autre sans repasser comme dans l'arbre au groupe précédent. Bien que cette architecture soit intéressante, elle ressemble fort à celle de la bibliothèque de Babel imaginée par J. L. Borges, dont l'étude n'a pas été prolongée. En effet, nous avons pensé qu'il serait plus judicieux de placer tous les livres dans une seule et même géométrie, afin d'encourager le lecteur à regarder d'autres livres que ceux du groupe de livres pertinents.

En voyant une illustration de la roue à livres de Ramelli[7] (voir la figure 4.13), nous avons alors imaginé une bibliothèque cylindrique en pensant plus particulièrement, au positionnement des livres en fonction de leur pertinence à la requête faite par l'utilisateur (voir le schéma de la figure 4.12). Nous nous sommes cependant heurtés à un problème pour placer des livres répondant à plus de deux critères. En effet, avec deux critères seulement, il est aisé de positionner les livres. Dans le schéma de droite de la figure 4.14, on place l'utilisateur dans le cylindre face aux livres pertinents et dos aux livres non pertinents. Lorsqu'il se déplace dans le cylindre vers le haut, il découvre des livres répondant uniquement au premier critère alors que s'il descend, il découvre les livres répondant au deuxième.

### *Dans une sphère*

Pour permettre de placer plus de groupes que dans un cylindre, nous avons réparti les livres de telle manière qu'ils occupent l'intérieur d'une sphère. Considérons une requête comprenant $n$ critères. La sphère dont le rayon est calculé en fonction du plus important des groupes est "découpée" en *n+1* parties à l'aide de l'angle d'azimut (coordonnées sphériques) de la façon suivante :
- Le groupe des livres pertinents (répondant aux $n$ critères) se trouve en azimut $-\pi/2$.
- Les groupes des livres répondant à *n-1* critères se trouvent en azimut $-\pi/2 + \pi/n$.

[7] Illustration extraite de "L'histoire de la lecture" de Mengel [Mengel98].

**Fig. 4.13** La roue à livres de Ramelli.

– etc.
– Les groupes des livres ne répondant qu'à $n$-$i$ des $n$ critères se trouvent en azimut $-\pi/2 + \pi * i/n$.
– etc.
– Le groupe des livres non pertinents (répondant à 0 critère) se trouve en azimut $\pi/2$.

Nous obtenons ainsi, non pas des fuseaux horaires, mais des fuseaux "nombre de critères pertinents". Puis, un second découpage est fait pour répartir uniformément ces groupes répondant à $i$ des $n$ critères de la requête, en utilisant l'angle de la hauteur. Pour cela, nous savons que leur nombre est de $c = C_n^i$ (nombre de combinaisons des $n$ critères pris $i$ par $i$) pour la i$^e$$me$ des tranches précédentes. Nous obtenons donc un des groupes de la i$^e$$me$ tranche tous les $2\pi/c$. Il est alors facile de calculer le point $P$ d'un groupe à l'aide des deux angles obtenus pour ce groupe et en fonction du rayon de la sphère.

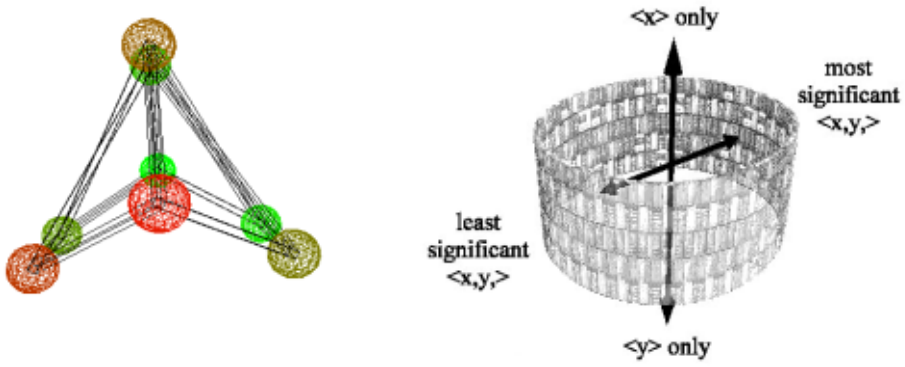


**Fig. 4.14** Interfaces en “atomium” et en “cylindre”.

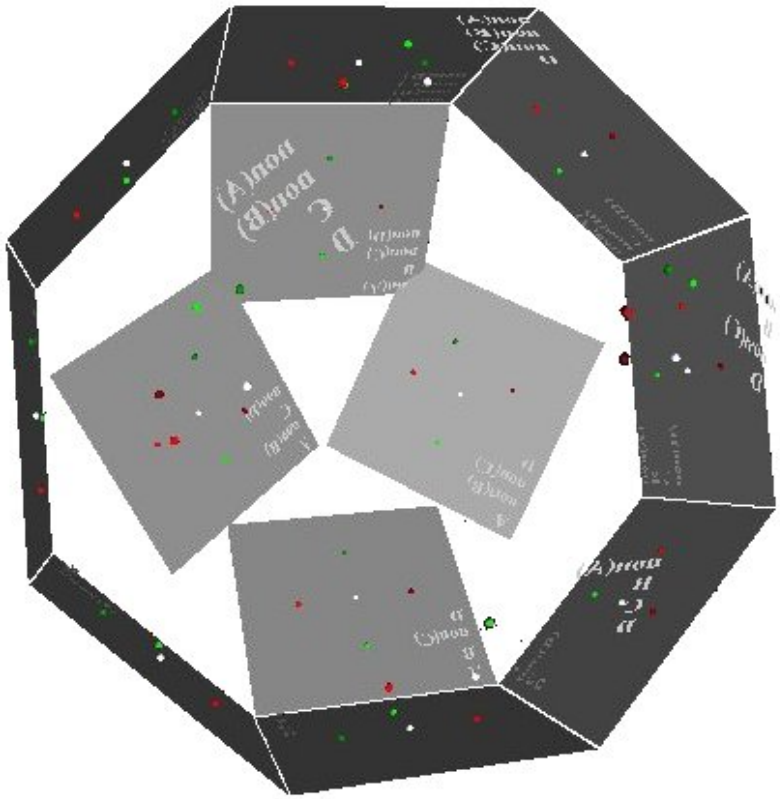

**Fig. 4.15** Placement des groupes de livres dans une sphère.

Les livres appartenant à un groupe doivent alors être couchés sur un quadrilatère reposant sur un plan passant par le point $P$ et perpendiculaire à la droite passant par $P$ et le centre $C$

de la sphère. Pour obtenir ce quadrilatère, nous calculons deux points supplémentaires. Le premier point ($A$) est obtenu en augmentant l'azimut du point $P$ de la valeur $\pi/(2(n+1))$. Le second point ($B$) est le résultat du produit vectoriel du vecteur $\vec{PA}$ avec le vecteur $\vec{CP}$. Nous obtenons ainsi deux points sur la sphère qu'il faut réorienter pour que $\vec{PA}$ donne la verticale et $\vec{PB}$ l'horizontale, lorsque l'utilisateur se trouve en $C$ et regarde vers $P$. Ceci est fait par une rotation des deux points de telle façon que $\vec{PA}$, après rotation, se trouve sur l'axe Z. La méthode, pour trouver la matrice de rotation, est expliquée dans le livre de Foley-van Dam [FOLEY90] à partir de la page 215. Le résultat obtenu est montré dans la figure 4.15. On y voit les emplacements occupés par les différents groupes de livres. Les livres peuvent alors être placés, un par un, en suivant les directions données par les deux vecteurs $\vec{PA}$ et $\vec{PB}$ de chaque groupe, les vecteurs donnant respectivement les directions des axes X et Y.

### 4.4.6 Interface de lecture

Nous présentons ici les deux prototypes réalisés pour permettre la lecture des ouvrages. Le premier a été construit d'une manière très naïve en pensant que des éléments 2D dans une interface 3D, pourraient aider l'utilisateur. Cependant, il s'est avéré que ce choix est très limitant et peu efficace. Néanmoins, cette étude nous a tout de même permis de tester les possibilités de VRML pour le rendu d'applications complexes. De cette étude, après avoir fait le constat que les scènes VRML scriptées sont trop complexes à mettre en œuvre, sont nées les premières extensions que nous exposons dans le chapitre 5.

Dans le deuxième prototype, nous proposons cette fois des objets 3D pour habiller notre scène. Par ailleurs, l'interface que nous présentons est nettement plus contrainte, facilitant ainsi grandement les interactions avec une souris 2D. Cette manière de contraindre les déplacements des objets et de la caméra, s'avère nettement plus efficace que de proposer des éléments non contraints, que l'utilisateur a l'habitude de manipuler en 2D, comme les fenêtres. Enfin, l'interface tire également profit de l'habillage 3D qui permet d'intégrer un plus grand nombre d'interactions.

#### Première génération en VRML

De manière à être consistante avec l'interface de localisation présentée précédemment et afin de tirer partie également des possibilités d'organisation offerte par la 3D, un poste de lecture 3D semblait intéressant à étudier. Le poste de consultation doit permettre la lecture des ouvrages sélectionnés lors de la phase de recherche. Cette étude essaye de dégager les interactions 3D nécessaires sur l'objet livre mais, également, de mettre à jour certaines insuffisances du langage VRML pour spécifier les comportements 3D interactifs génériques. Ce deuxième point s'avère être un problème général aux langages de description des scènes 3D.

### *Les choix généraux*

Le point important dans cette interface est de permettre aux utilisateurs d'organiser leur espace de travail, d'agencer les livres numérisés dans l'interface, de la même manière que les *webbooks* peuvent être organisés dans le *Web Forager* [CARD96]. Pour faciliter ces interactions, nous avons présenté les éléments à manipuler dans des fenêtres 3D (c'est-à-dire des fenêtres 2D sur lesquelles on interagit en 3D). Cela permet de profiter du savoir-faire qu'ont acquis les utilisateurs pour interagir sur les fenêtres 2D. Ils connaissent parfaitement l'utilité de la barre de titre et la signification des boutons. Il ne leur reste plus qu'à se familiariser avec les déplacements possibles d'une fenêtre dans un espace 3D.

La scène initiale est constituée d'une seule fenêtre comportant les ouvrages sélectionnés. Pour l'instant, puisqu'il n'y a aucune liaison entre l'interface de localisation et le poste de lecture, ces livres sont tous les ouvrages consultables depuis le CNUM. Ils ne correspondent pas à une sélection faite dans l'interface de recherche. Tous les livres sont représentés par leur dos, offrant là encore des informations visuelles telles que la date d'édition du livre grâce à son type de reliure, le nombre de pages selon la largeur de la reliure ou encore le nombre total d'ouvrages.

Le moteur 3D utilisé pour cette expérimentation est celui géré par tout *plugin* VRML. En effet, nous utilisons le langage VRML pour décrire notre interface. Les interactions, quant à elles, sont spécifiées en Java, un des langages de scripts permettant d'augmenter les possibilités de VRML. Notre scène peut, par conséquent, être dynamique (création, modification et suppression de contenu) grâce à ce couple VRML+Java. Elle est donc visualisable sous n'importe quelle application prenant en charge ces deux langages : pratiquement tous les *plugins* VRML. L'utilisation de VRML comme langage support est une solution aussi valable que n'importe quelle autre. Ce qui peut paraître plus étonnant en revanche, c'est d'utiliser un *plugin* VRML comme moteur 3D de notre application à la place d'une API 3D. Cependant, cette approche offre quelques avantages :
- la bibliothèque est accessible depuis Internet,
- aucune installation n'est nécessaire pour lire les ouvrages de la bibliothèque, hormis celle d'un *plugin*,
- l'utilisateur emploie son *plugin* préféré, dont les outils d'interactions qu'il connaît bien, le satisfont.

De plus, l'utilisation d'un *plugin* VRML, comme moteur 3D, permet à l'utilisateur d'évoluer de deux manières différentes dans notre scène. La première se fait par l'intermédiaire des interacteurs disposés dans la scène. Elle s'adresse aux utilisateurs novices, ne connaissant pas bien les métaphores de navigation 3D. La deuxième, plutôt pour les experts qui la verront comme un complément à la première, s'appuie sur les outils de navigations standards d'un *plugin* : *walk*, *pan*, *study*, etc.

Quant aux avantages d'une solution en 3D plutôt qu'une interface 2D, ils découlent des considérations simples et courantes de la 3D :
- utilisation d'un volume théoriquement infini, au lieu d'une surface fixe, pour représenter les éléments de l'interface,
- utilisation de la profondeur pour détailler les informations (offrir un historique, afficher des informations dont la structure est véritablement 3D) ou les organiser,

– utilisation de la 3D pour son pouvoir de représentation.

D'autres avantages, non nécessairement propres à la 3D, sont néanmoins présents et fortement utilisés :

– la transparence qui permet d'apercevoir les éléments se trouvant derrière un autre,
– le point de vue de l'utilisateur sur les données qui peut changer (et en particulier un zoom est naturellement implémenté).

Pour la consultation de documents, la 3D offre une solution au problème engendré par l'utilisation de fenêtres "recouvrantes" (*overlapping windows*). Ces fenêtres, généralement utilisées dans les interfaces graphiques "modernes", sont gérées comme une pile. Dans la plupart des cas, seule la première est visible, les autres étant soit recouvertes (partiellement ou entièrement) soit iconifiées. La figure 4.16 montre l'interface de lecture 2D actuelle du CNUM, juxtaposée au prototype d'interface 3D. Les documents sont ouverts et mis à l'échelle souhaitée et l'utilisateur sait toujours où ils se trouvent grâce à la transparence. Dans une interface 2D, plusieurs fenêtres pourraient être ouvertes de la même manière mais la navigation de l'une à l'autre serait fort désagréable. Ici, les déplacements des fenêtres sont fluides, autorisant ainsi le lecteur à utiliser sa mémoire spatiale.

### *Détails d'implémentation*

Puisque nous désirons utiliser des fenêtres que le lecteur puisse ouvrir et fermer à sa guise, il faut un mécanisme permettant de créer et d'enlever du code VRML à la scène initiale. La seule possibilité consiste à utiliser les méthodes *CreateVrmlFromString* de la classe *Script* et *destroy* de la classe abstraite *Vector* qui nous sert à énumérer toutes les fenêtres. Le code du fichier VRML principal ne contient aucune définition d'objets mais uniquement les nœuds suivants :

– *Group* contenant toutes les fenêtres,
– *ProximitySensor* servant à inhiber les transformations de la caméra pour les fenêtres punaisées,
– *Script* définissant la classe Java principale *MainScript*, gestionnaire de la scène.

La méthode *initialize*, lancée à l'initialisation de ce script (donc à celle de la scène) construit le code VRML de la première fenêtre en fonction des livres présents dans la bibliothèque CNUM. Un prototype de livre est utilisé. Il définit son apparence et déclare le script *BookScript* qui gère les interactions sur ce livre. Lorsqu'un clic de souris a lieu sur le dos d'un livre, un événement est propagé vers ce script qui demande la création d'une fenêtre supplémentaire au gestionnaire de fenêtre. Le code VRML des objets constituant les informations de ce livre est inséré dans la nouvelle fenêtre. Les boutons de consultation d'un livre sont gérés de la même manière : ils émettent un événement qui change le mode de la fenêtre (passage du mode notice au mode lecture) et qui crée l'interface de lecture gérée par la classe *PageScript*.

Les fenêtres 3D sont prises en charge par la classe *Window3D* et leur code VRML est généré dans la méthode *create*. Ce code contient un nœud *Script* déclarant l'utilisation de la classe *Window3Dscript* qui sert à récupérer les interactions sur les fenêtres. Il est chargé également de calculer la transformation inverse et de l'appliquer à la fenêtre si elle doit rester immobile. Les transformations du point de vue sont récupérées grâce au nœud

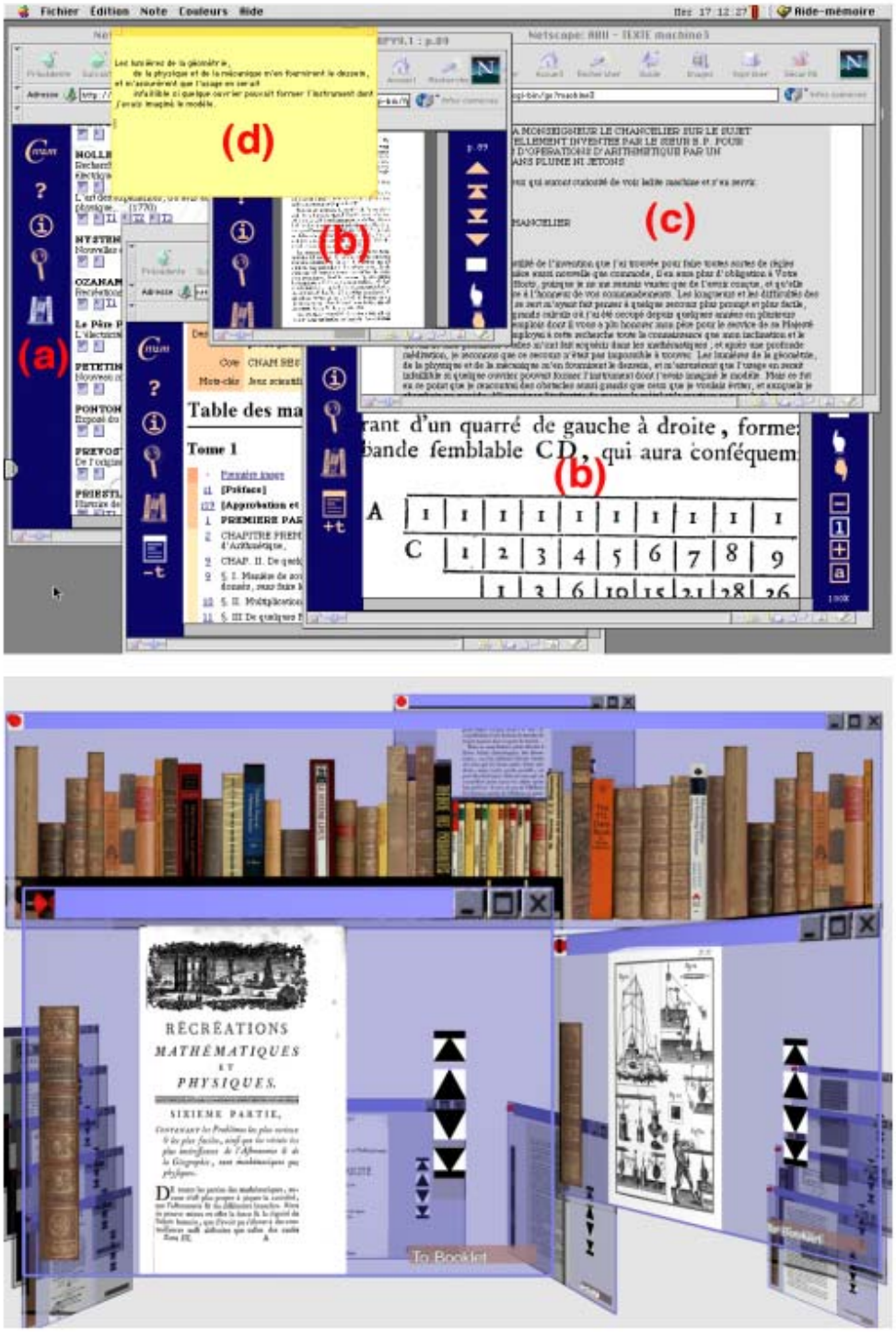


**Fig. 4.16** Les interfaces de consultation du CNUM en 2D et en 3D.

*ProximitySensor* dont la boîte définissant ses frontières de fonctionnement est arbitraire-

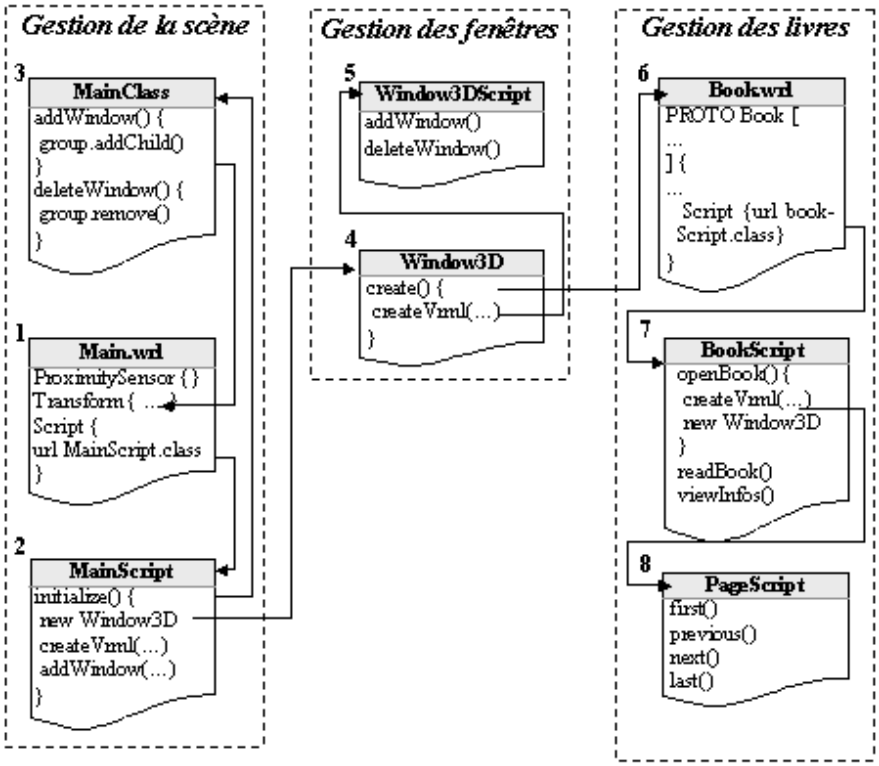


**Fig. 4.17** Architecture du poste de lecture.

ment grande afin qu'il émette toujours les changements. Les déplacements d'une fenêtre se font par "glisser-déposer" en cliquant sur sa barre de titre. Ils sont réalisés par un routage de l'événement sortant *translation_changed* du nœud *PlaneSensor* (qui renvoie les déplacements de la souris dans le plan de la barre de titre) vers l'attribut *set_translation* du nœud *Transform* contenant toute la fenêtre.

L'architecture logicielle décrite est schématisée dans la figure 4.17. Bien que les comportements soient simples, ils représentent plus de 1200 lignes de code en Java pour 160 lignes de VRML, soit la quasi-totalité de la scène téléchargée.

### *Extension pour la vidéo*

Ces mêmes scripts nous ont également servi pour construire un prototype de vidéothèque 3D. Ce projet a été présenté à la Direction Technique du Ministère de la Recherche et des Technologies (MRT/DT) comme candidat possible pour accéder aux émissions et vidéos de la chaîne universitaire Canal U.

S'agissant d'un prototype, nous n'avons pas intégré les vidéos en *streaming* mais uniquement les formats pris en charge par le nœud *MoviePicture* de VRML (les formats AVI et

MPEG). Le *plugin* VRML Cortona de ParallelGraphics propose cependant une extension pour utiliser des vidéos au format *Real Video*.

La mise à jour du poste de lecture en visionneuse de films n'a pas demandé beaucoup de travail. Il a en effet suffit de construire une base de données de films contenant les auteurs, les titres, ainsi que d'autres informations utilisées pour la fiche signalétique et de fournir l'URL vers le film. Dans le PLAO (Poste de lecture Assisté par Ordinateur), les URL vers les images du CNUM sont utilisées pour renseigner le champ *url* des nœuds *ImageTexture*, afin de texturer une face rectangulaire et ainsi de simuler une page de livre. De la même manière, dans la visionneuse de films, les URL des films renseignent le champ *url* des nœuds *MovieTexture*. La face texturée avec une vidéo devient alors un écran de visualisation.

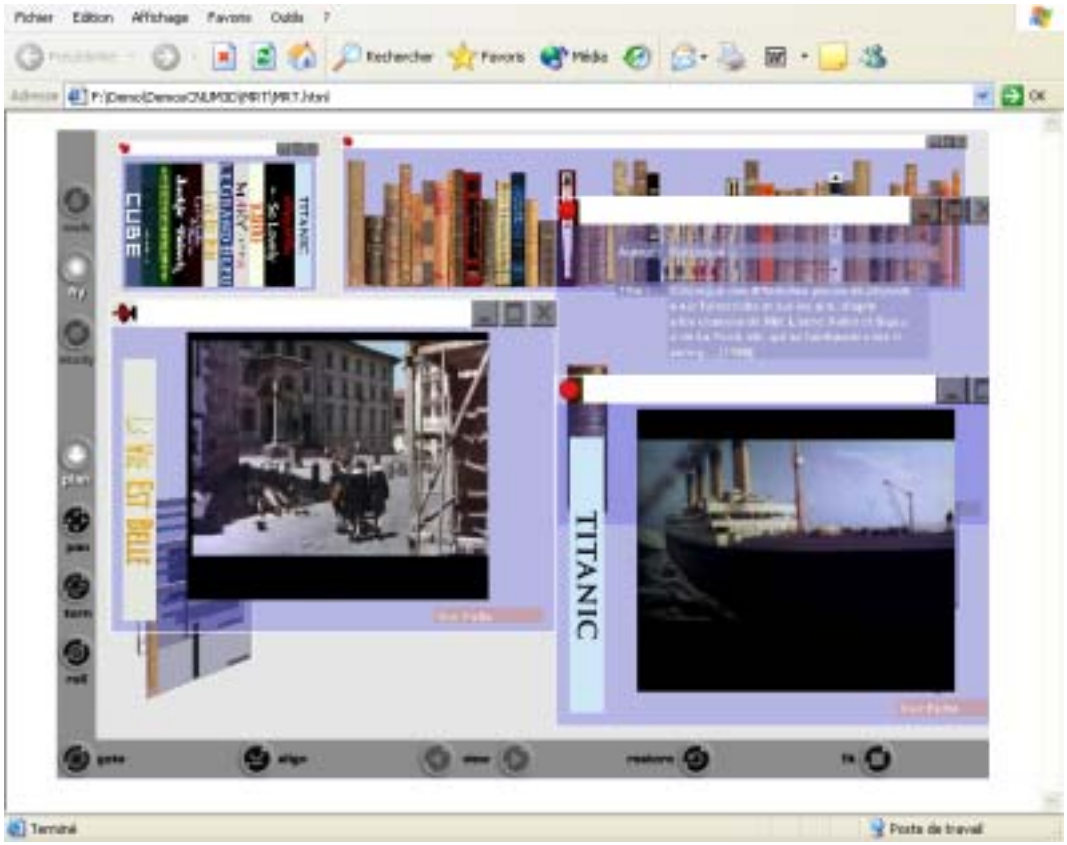


**Fig. 4.18** Notre prototype de médiathèque.

La figure 4.18 est une copie d'écran de notre médiathèque (car elle inclut une bibliothèque et une vidéothèque). Y sont représentées deux collections ; l'une est constituée des livres du CNUM (en haut à droite) et l'autre comprend des extraits de films (en haut à gauche). Pour cette deuxième collection, nous avons conservé les mêmes principes de représentation et d'interaction que pour la première. On peut voir, par exemple, l'inclusion de la jaquette des films pour distinguer facilement un film d'un autre. Ces informations visuelles, bien que

précieuses, sont toutefois moins nombreuses que dans la collection de livres. Les dimensions des jaquettes sont en effet toujours constantes contrairement à celles des livres.

Dans cette interface, se mêlent les fenêtres ouvertes sur des livres ou sur des films. Elles ont les mêmes propriétés et subissent les mêmes interactions. Dans les deux cas, l'utilisateur aura la possibilité de visualiser la fiche du média ou le média lui-même. On peut encore augmenter facilement ce modèle en proposant des enregistrements audios spatialisés grâce aux nœuds *Sound*. Il faudra, en fonction de la fenêtre active, déterminer les sons jouer et ceux à arrêter afin d'éviter les cacophonies.

### *Critiques de l'interface*

Tous les éléments dans l'interface sont en deux dimensions et peuvent donc être insérés dans des fenêtres 3D transparentes. L'utilisateur peut agir directement sur les fenêtres pour les déplacer, les icônifier, les maximiser, les punaiser ou les fermer. Ses interactions issues des interfaces 2D, sont bien connues et devraient satisfaire les utilisateurs novices qui n'ont pas ou peu d'expérience de manipulation d'objets en 3D. Ces utilisateurs peuvent même consulter les documents sans déplacer le point de vue sur la scène. Ils bénéficient, néanmoins, du rangement des fenêtres en perspective grâce au bouton d'icônification.

Des objets 3D auraient pu être une meilleure solution pour obtenir un effet visuel plus agréable. Une métaphore de livre serait sans aucun doute plus pratique et compréhensible au premier abord. Les interactions avec cet objet pourraient être également beaucoup plus poussées. Par exemple, un outil pourrait être fourni pour feuilleter le livre afin d'appréhender la structure du livre ou retrouver un passage particulier grâce à la mémorisation photographique de certaines pages que l'on conserve. Cependant, ces interactions supplémentaires nécessiteraient de surajouter des *widgets* supplémentaires à côté du livre, pour les activer. Soit l'apparence du livre en serait modifiée (et n'aurait plus rien de réel), soit les *widgets* apparaîtraient lorsque le livre est activé. Nous ne nions pas le fait qu'une telle interface aurait une plus grande puissance expressive. Cependant, dans le contexte de cette étude, le but recherché était d'offrir une interface intuitive. C'est pourquoi nous nous sommes appuyés sur des mécanismes d'interaction directement accessibles et facilement compréhensible, car issus des interfaces 2D.

Quel que soit l'élément sur lequel interviennent les interactions, l'apparence de l'objet est modifiée la plupart du temps. Cela signifie que le graphe de scène VRML est modifié par les classes Java. Or, pour que les classes connaissent la sémantique des éléments du graphe, elles doivent le construire. De ce fait, les scripts représentent plus de 90% du code total de la scène. Ce couple VRML+Java peut être comparé à l'API Java3D puisque la syntaxe du langage VRML nous permet de décrire le graphe de scène, et le langage Java est utilisé pour le modifier. L'apport de VRML en tant que langage support est, dans ce cas, négligeable. Et cela pour une simple et unique raison : l'absence de comportements au niveau de la description des scènes.

C'est uniquement une voie que nous proposons pour résoudre ce problème et que nous envisageons d'étudier plus tard. Quelques nœuds de comportements simples permettraient de réduire la part des scripts et donc également l'effort de programmation nécessaire pour implémenter une telle scène. D'autre part, ces nœuds autoriseraient des non informaticiens

à produire des scènes interactives. Dans le même ordre d'idée, les modeleurs spécialisés pourraient alors exporter facilement, des scènes interactives. Les problèmes de confidentialité du code Java, qui représente tout le savoir dans ce genre de scène et qui est facilement consultable grâce à un traducteur de classe en code source Java, seraient aussi réglés. Ces considérations sur les problèmes posés par une telle solution, ont été présentées à la conférence ERGO-IHM'2000 [CUBAUD00]. Quant à l'architecture de notre prototype, elle a été présentée à la conférence Web3D'2001 [CUBAUD01].

### Deuxième génération avec Virtools Dev

L'idée principale de cette deuxième étude est de proposer de nouvelles métaphores pour accéder aux services offerts par les bibliothèques numériques. Bien qu'ayant voulus profiter des acquis des utilisateurs pour manipuler les fenêtres 2D, notre premier prototype souffrait d'une interface qui ne mettait pas à contribution les possibilités offertes par une représentation 3D. Nous nous sommes donc avant tout attachés à repenser les éléments nécessaires à la manipulation et à la lecture d'un fonds. Dans un deuxième temps, nous avons imaginé la façon d'intégrer l'interface de recherche dans le fonds et le poste de lecture, dans un même environnement. Les différentes métaphores qui nous sont apparues sont présentées dans la section 4.4.7 à partir de la page 169. Ces études ont été réalisées durant le stage de DEA de Pascal Stokowski et ont abouti à la présentation de deux articles, l'un à la conférence JCDL'2002 [CUBAUD02A] et l'autre à VRIC'2002 [CUBAUD02B].

Nous nous sommes très vite aperçus que plusieurs problèmes se posaient, avec notre premier prototype du poste de lecture. Tout d'abord, il y a discontinuité avec l'interface de recherche puisque les paradigmes de représentation et d'interaction ne sont pas les mêmes. On utilise des objets 3D dans l'interface de recherche et des conteneurs 2D dans le poste de lecture. De plus, la navigation et l'interaction parmi les fenêtres 2D ne sont difficiles à cause du trop grand nombre de degrés de liberté. Or, ce manque de contraintes dans les déplacements de la caméra et des objets, nuit à la simplicité de l'interface.

Il nous est donc naturellement venu à l'esprit d'habiller les livres d'un conteneur 3D et de limiter les interactions possibles pour simplifier les déplacements. Notre axiome de base a été de fixer les mouvements possibles à 4 degrés de liberté (deux translations et deux rotations) pour qu'ils conviennent à l'utilisation d'une souris 2D. Pour cela, nous avons défini un sol sur lequel les objets devaient être posés verticalement. Par cette simple contrainte, nous avons éliminé les translations selon l'axe (vertical) Y (impossibilité de s'élever au dessus du sol) et les rotations selon l'axe X (impossibilité d'incliner la scène).

Alors que nous présentions notre premier prototype de poste de lecture à la conférence Web3D'2001, nous avions déjà réfléchi sur la suite à donner à ce projet. L'ébauche en POV (*Persistance Of Vision*) de l'interface du deuxième prototype était déjà effectuée, et a d'ailleurs été sélectionnée pour apparaître sur la couverture des actes de la conférence (voir la figure 4.19). Dans cette première ébauche d'interface métaphorique, l'élément le plus important est, sans conteste, le trépied que nous avons étudié plus en détail, par la suite. Néanmoins, nous présentons tous les nouveaux aspects de chacun des éléments, constituant l'interface de ce prototype.

**Fig. 4.19** La première ébauche du deuxième prototype de poste de lecture.

La maquette réalisée (et proposée en copie d'écran dans la figure 4.20) n'est pas aboutie mais offre cependant quelques fonctionnalités intéressantes. La gestion du point de vue de l'utilisateur est modifiable dynamiquement grâce au clavier. Les ouvrages sur le sol peuvent être manipulés librement. Les pages peuvent être feuilletées sur certains trépieds. Il est également possible de changer l'échelle d'affichage des documents. Il reste bien entendu beaucoup de travail à effectuer mais l'ensemble de cette réalisation nous a permis de constater que Virtools Dev est un outil qui convient à l'implémentation, tout du moins au prototypage rapide, du poste de lecture. Nous avons pu, par exemple, tester la capacité du *plugin* à supporter les nombreuses textures. Dans la copie d'écran de la figure 4.20, chacun des livres affichés est habillé d'une texture unique (nous avons pour cela dupliqué des fichiers d'image).

### *Le livre*

Le point le plus important concernant la présentation des ouvrages est la possibilité de les parcourir efficacement. Pour cela, outre la manière classique (pages suivante et précédente, première et dernière pages), nous devons permettre le feuilletage, de la même manière que les *webbooks* sont feuilletés dans le *Web Forager* [Card96]. Nous avons également intégré des marques-pages, plutôt que de reproduire l'autre solution implémentée dans le *Web Forager*, c'est-à-dire une loupe à documents (voir page 69). Les signets, introduits dans les pages du livre, déclenchent l'affichage d'une bulle d'aide (comportant le chapitre, le numéro de la page, des annotations, etc.) lorsqu'on les survole avec la souris (*Roll Over*). Un double clic permet d'afficher la page correspondante.

**Fig. 4.20** La maquette sous Virtools du poste de lecture.

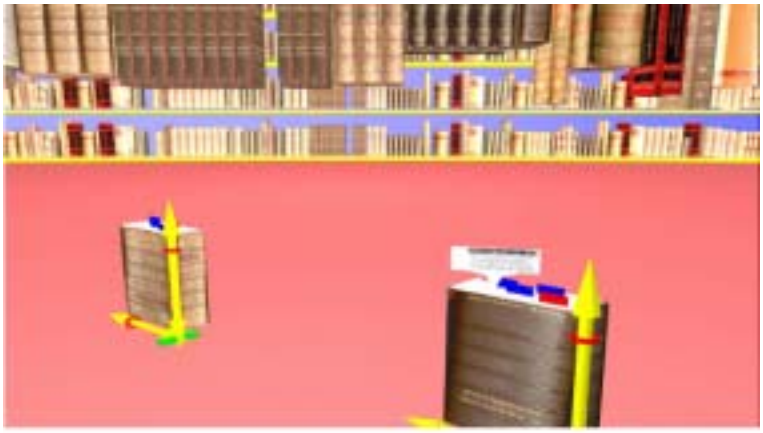

**Fig. 4.21** Le livre et ses marques-pages.

Comme dans notre premier prototype, les dimensions des ouvrages sont proportionnelles à leur taille réelle : ceci afin de fournir à l'utilisateur des informations visuelles complémentaires. Pour la même raison, nous utilisons la couverture et la dos du livre, pour habiller la boîte qui le représente dans l'environnement 3D.

### *Le trépied*

Le croquis reproduit dans la figure 4.22 est la première proposition faite par Pascal Stokowski pour habiller le livre. Plus exactement, le trépied, imaginé par Pierre Cubaud, a été adapté en prenant en compte la liste exhaustive des interactions nécessaires sur l'objet livre. Le trépied est ainsi devenu le composant par lequel on manipule principalement l'objet livre.

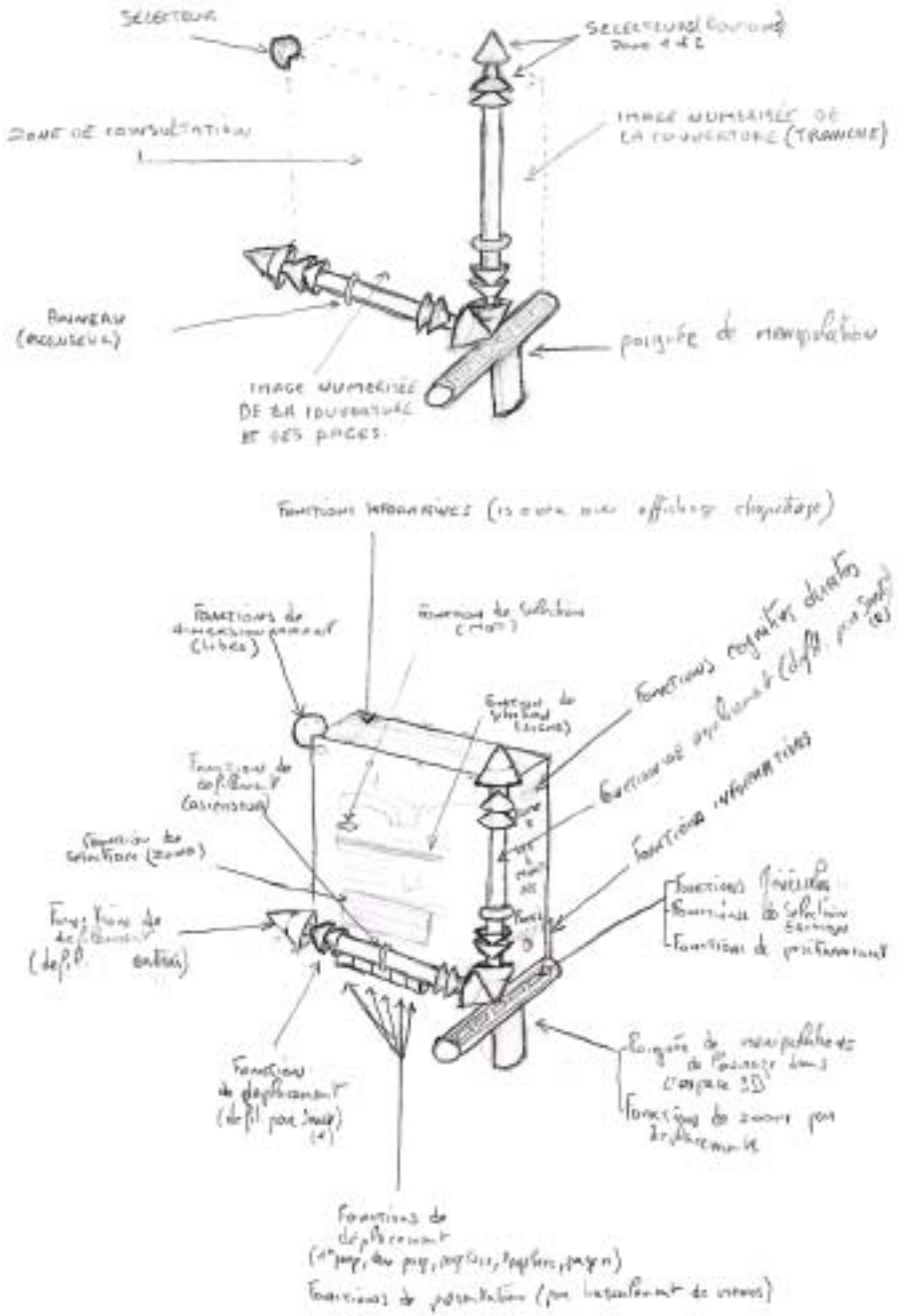


**Fig. 4.22** Les fonctionnalités du trépied.

Un certain nombre de fonctionnalités que l'on peut étudier en détail dans [Stokowski02], ont donc été intégrées au trépied :
– recherches d'occurrences,
– édition ou annotation,
– opérations sur les pages du livre (défilement, feuilletage, zooms, ...),

– rangement, dimensionnement, déplacement,
– présentation (page unitaire ou double page, changement du point de vue, ...).
L'ouvrage ne servira qu'à l'affichage des pages et ne sera jamais caché par un *widget*. Le trépied, quant à lui, intègre toutes les opérations sur le livre. Dans le mode lecture, ces deux objets sont indissociables. La représentation correspondante du trépied en 3D est visible dans la figure 4.23.

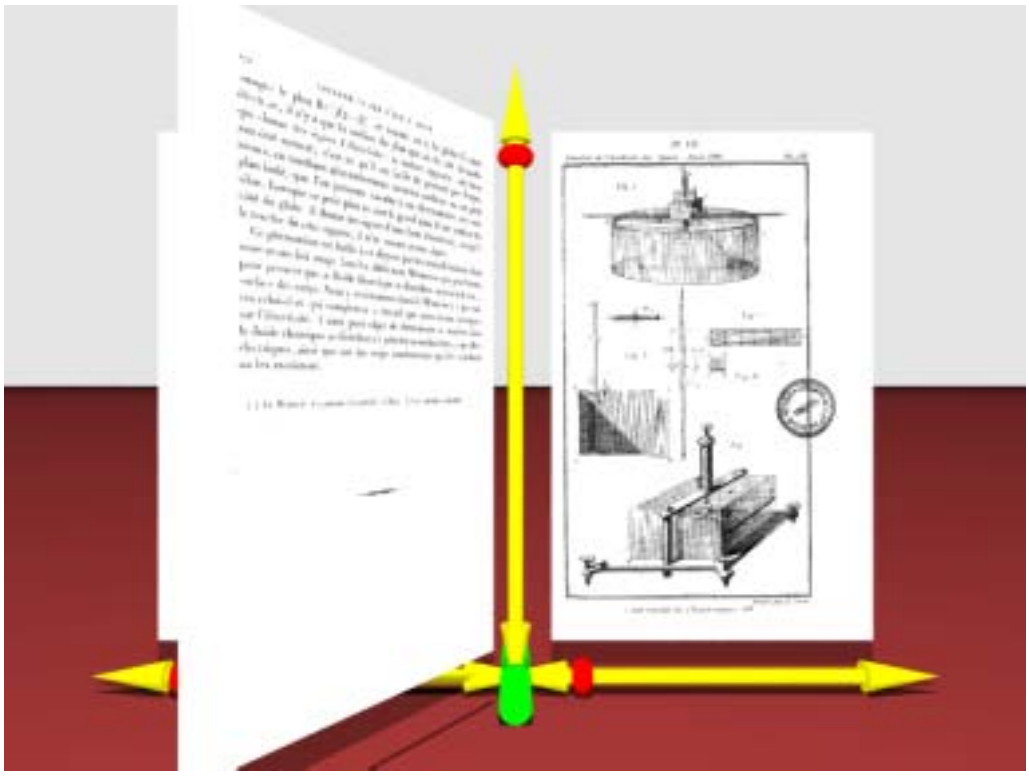

**Fig. 4.23** La métaphore du livre 3D avec son trépied.

### *Le sol et l'arrière plan*

Le sol, absent dans nos précédents prototypes, revêt une grande importance dans celui-ci. Il joue le rôle de support et c'est donc lui qui contraint les objets à n'utiliser que quatre degrés de liberté. Par ailleurs, il définit une manière simple et efficace pour appréhender la profondeur des objets grâce aux ombres qui y sont projetées. Une surface quadrillée permet également d'appréhender facilement la profondeur. Notre prototype final devra proposer les ombres portées sur un sol dallé.

L'arrière plan, quant à lui, n'a d'importance que pour apaiser les yeux de l'utilisateur. Un fond uni ayant une couleur douce est en effet plus agréable qu'un fond blanc ou noir, l'une des deux couleurs généralement utilisée comme couleur de fond par défaut.

### *Le panier de sélection*

Le panier est la métaphore choisie pour passer de l'interface de recherche à l'interface de lecture. C'est le seul élément qui soit commun aux deux interfaces et c'est par lui que se fera le lien. Dans l'interface de recherche, les ouvrages sélectionnés pour être lus ultérieurement, sont positionnés dans ce panier. Une fois dans l'interface de lecture, le panier représente et conserve une sous-collection accessible directement, sans retourner à l'interface de recherche. Un panier en transparence est illustré dans la figure 4.24.

**Fig. 4.24** Une représentation du panier au premier plan.

Le panier offre un moyen efficace pour accéder aux livres et pour les ranger. Cet élément doit être toujours visible et accessible, car c'est sur lui qu'interviendront les interactions permettant de basculer de l'environnement de recherche au poste de lecture. D'autre part, il est particulièrement important que les ouvrages sélectionnés par l'utilisateur, dans l'interface de recherche, soient toujours accessibles et facilement consultables, puisque tel est le but du poste de lecture. Le survol des ouvrages permet l'affichage dynamique d'informations textuelles (références, résumé, table des matières ou des illustrations, etc.). Un double clic sur un ouvrage l'ouvre en le présentant au premier plan, habillé de son trépied.

L'apparence actuelle du panier n'est pas encore son apparence définitive. Nous pensons plutôt proposer un panier ressemblant aux menus 3D ou au *dock* de MacOS X dans lesquels sont mis en œuvre des rotations ou des homothéties pour faire ressortir l'élément en passe d'être sélectionné.

### 4.4.7 La liaison entre les deux interfaces

La sélection d'ouvrage dans une collection et la lecture des livres sélectionnés dans une interface prévue à cet effet, sont deux tâches séparées dans la plupart des bibliothèques numériques. Avec nos interfaces 3D aussi, il y a une nette séparation entre la recherche et la lecture. Le changement de contexte induit par cette séparation des deux interfaces provoque une perte de temps notable lors de la navigation et lors de l'apprentissage. C'est pourquoi nous avons étudié, pour nos interfaces, de quelle façon les métaphores 3D sélectionnées peuvent nous donner une navigation continue entre les deux interfaces.

Deux options peuvent être considérées pour mettre en commun la recherche et la lecture de livres. La première est de mélanger le contenu des deux interfaces dans une seule interface. La seconde est de les séparer dans l'espace et d'utiliser une métaphore 3D pour passer de l'une à l'autre. Quelle que soit la solution choisie, il est important de fournir les mêmes métaphores de navigation et d'interactions. De cette manière, les deux outils mis en commun, ne poseront pas de problèmes et ne gêneront pas l'utilisateur qui travaille avec la collection et les livres.

Dans le cas où, pour implémenter une métaphore particulière, il y a impossibilité de placer les outils avec une égale importance, nous préférons favoriser la collection. Nous considérons en effet que la collection est l'élément principal de la bibliothèque, puisque le travail et l'intérêt de l'utilisateur en dépend. L'approche opposée est une interface centrée sur le document, comme l'*Information Visualizer* [Card91] ou le *Web Forager* [Card96], dans lesquels les outils sont produits à la demande, pour aider l'utilisateur à travailler sur les documents.

Dans toutes les métaphores proposées, la navigation et l'interaction avec les documents sont contraintes pour faciliter l'accès à la bibliothèque 3D pour les novices. Cependant, nous fournissons assez d'outils pour que toutes les opérations essentielles soient permises :
- interrogation de la base de données,
- parcours de la collection générée en fonction de la requête,
- sélection, pré-visualisation des livres,
- lecture des livres.

La pré-visualisation des livres utilise la même représentation, par utilisation du trépied, que la lecture des livres. Les fonctions incluses dans le trépied sont toutefois moins nombreuses. Dans la partie collection, les interactions forment donc un sous-ensemble des interactions possibles dans l'interface de lecture. La mise en commun de ces environnements est donc faisable. Il suffit, pour cela, de fournir une transition simple pour passer de l'un à l'autre. Afin de ne pas surcharger l'interface, nous préférons ne pas utiliser de *widgets* pour conserver le même type d'interactions dans toute l'interface. Les métaphores d'intégration des deux tâches, de recherche et de lecture, ont été scénarisées en utilisant les scripts de POV. Nous ne fournissons dans ce rapport, que des copies d'écran de ces animations[8].

Dans les trois métaphores proposées, le seul élément commun est le panier contenant les livres présélectionnés. Il est toujours visible et c'est le lien entre les deux interfaces.

[8] Les animations complètes sont disponibles sur notre site web : `http ://cnum.cnam.fr/3D`.

## La métaphore du "cockpit"

La collection est présente en arrière plan. Le panier et les livres ouverts sont représentés au premier plan, devant la collection. Dans cette métaphore, tous les éléments partagent donc le même espace. Les livres et le panier ne subissent pas les mouvements de l'utilisateur, seule la collection la subit. D'autre part, les éléments au premier plan sont toujours visibles.

Le nom donné à cette métaphore vient de la manière dont les données sont visualisées. Un pilote de chasse dans son *cockpit*, voit non seulement le paysage mais également les données qui lui sont soumises en surimpression dans son casque. Ces données sont donc toujours visibles et ne subissent pas les mouvements de tête du pilote contrairement au paysage. De la même manière, dans notre interface, la collection forme le paysage et, les livres ouverts et le panier, les données persistantes (voir figure 4.25).

L'inconvénient de cette métaphore est la surcharge de l'écran avec toutes les données. Une attention particulière doit donc être apportée pour fournir une résolution maximale pour les deux tâches. Donc, la séparation géographique des deux environnements ou tâches dans une même interface semble, pour cette raison, un bon choix. C'est le cas notamment des deux interfaces que nous présentons par la suite.

**Fig. 4.25** La métaphore du "cockpit".

### La métaphore du "camembert"

L'espace est séparé en deux zones horizontales ayant une taille variable comme on peut le remarquer dans la figure 4.26. L'utilisateur peut basculer d'une tâche à l'autre en uilisant une rotation autour de l'axe Y, comme cela est schématisé dans la figure. La collection est affichée dans la première zone. Dans celle-ci, l'utilisateur peut modifier son point de vue pour aller à la rencontre des livres qu'il recherche.

Dans la seconde zone, les livres ouverts sont affichés sur le sol. L'utilisateur y a un point de vue fixe mais il peut, par contre, manipuler les livres. Afin de simuler l'immobilité de cette deuxième zone, lorsque l'utilisateur agit sur la collection, les livres et le sol doivent être déplacés pour compenser le mouvement dans la collection. De cette manière, quand l'utilisateur se retourne, il a l'impression que rien n'a bougé derrière lui. Il peut ainsi utiliser sa mémoire spatiale et se souvenir de la position des ouvrages. Les deux zones sont complètement séparées pour qu'il n'y ait pas de recouvrement. Seul le panier est présent dans les deux.

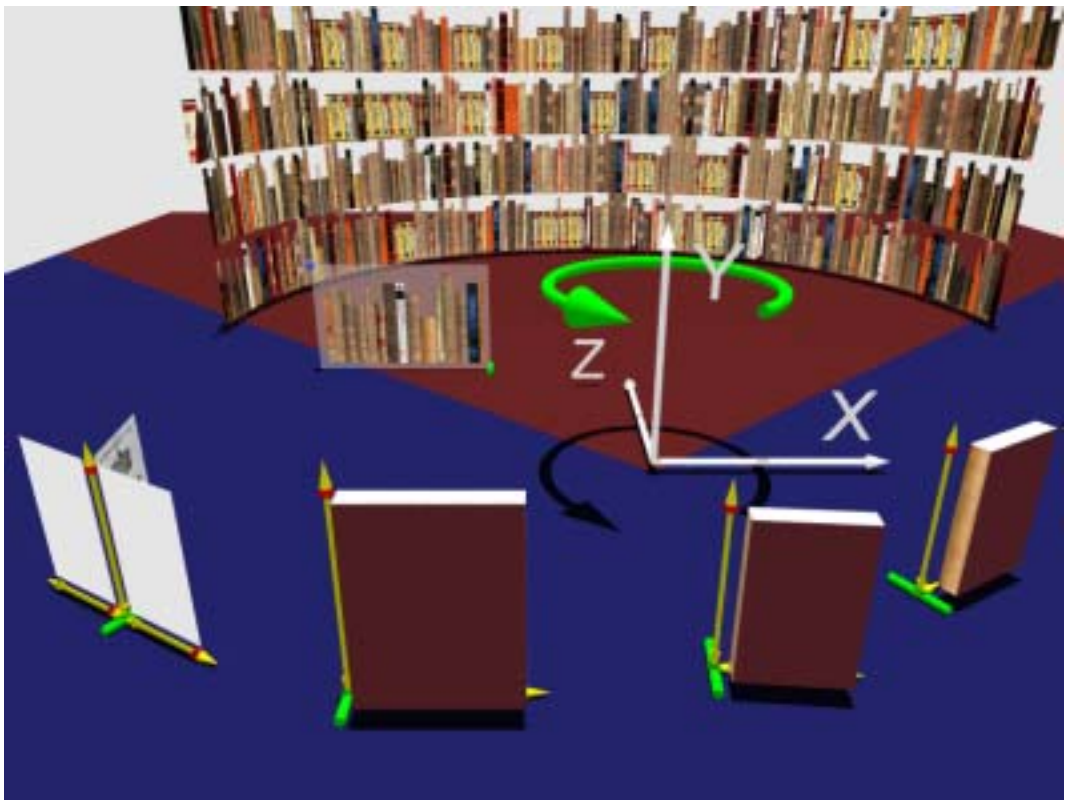


**Fig. 4.26** La métaphore du "camembert".

Avec cette métaphore, l'utilisateur peut créer plusieurs environnements de lecture. Elle prend ainsi, dans ce cas, tout le sens du nom qu'on lui a donné. Les différents environnements pourront avoir des tailles différentes et se répartiront l'espace horizontal laissé

vacant par la collection. En adaptant les tailles des zones de lecture à l'angle de vue de la caméra, il sera possible de simuler la séparation complète des différentes zones de lecture.

### La métaphore "sans dessus-dessous"

Le problème principal avec la métaphore précédente est que certains livres peuvent être occultés par la collection à cause de la compensation de mouvements. Après un certain nombre de déplacements dans la collection et, en particulier, après s'être déplacé en profondeur, les livres peuvent disparaître de la zone de lecture. On peut toutefois contraindre les livres à rester dans la zone de lecture. Cependant, dans ce cas, lorsque l'utilisateur se retournera, il ne reconnaîtra pas l'environnement de lecture qu'il avait laissé précédemment. De plus, la métaphore du "camembert" n'utilise que la moitié de l'espace disponible (au-dessus du sol).

Pour régler ces deux problèmes, nous pouvons séparer l'espace en deux zones verticales plutôt qu'en deux zones horizontales. Le déplacement du point de vue dans la collection n'affectera ainsi en rien l'environnement de lecture. Aucune compensation de mouvements et donc, aucun calcul supplémentaire, ne sera nécessaire. Pour basculer d'une activité à l'autre, il suffira cette fois-ci d'effectuer une rotation selon l'axe X ou l'axe Z, comme cela est schématisé dans la figure 4.27. Cette figure, ainsi que la précédente, n'illustre pas exactement ce que l'utilisateur voit. En effet, celui-ci est immergé dans la scène et ne peut pas observer les deux environnements en même temps. Le recul du point de vue, dans ces figures, n'est fourni que pour comprendre la structure globale des interfaces.

Dans la zone de lecture, l'utilisateur peut toujours créer de multiples environnements (plus nombreux que précédemment puisqu'il dispose de 360 degrés). La rotation autour de l'axe Y sera utilisée exclusivement pour le passage d'un environnement de lecture à un autre.

## 4.4.8 Suite à donner à ces études

Les métaphores décrites ci-dessus ne sont pas encore implémentées dans un système temps réel. Elles ne sont que décrites dans des scripts en POV. Sachant que la création d'une bibliothèque numérique en VRML, et plus généralement de tout environnement très interactif, est laborieuse et difficile à cause du nombre de scripts nécessaires, nous lui préférons d'autres solutions. Parmi les solutions possibles, nous envisageons l'utilisation d'une API 3D ou un outil de haut niveau comme Renderware, le moteur de jeu vidéo de chez Criterion qui s'appuie sur Direct3D ou OpenGL.

Cependant, sachant que la bibliothèque numérique doit être accessible en ligne, nous envisageons plutôt Java3D qui peut être intégrée dans une applet. Une autre solution peut être d'utiliser le langage VRML augmenté que nous présentons dans le chapitre suivant. Il permet en effet de réduire considérablement le nombre de scripts requis en proposant des nœuds génériques pour les interactions. Ce langage VRML augmenté n'est cependant interprétable que par nos navigateurs VRML spécifiquement développés pour cela. On peut imaginer néanmoins un outil permettant de traduire les nouveaux nœuds de notre langage VRML augmenté, en scripts Java exécutables depuis n'importe quel navigateur VRML.

**Fig. 4.27** La métaphore "sans dessus-dessous".

La solution la plus intéressante, dans ce contexte, est probablement l'utilisation de l'outil Virtools Dev que nous avons présenté dans la section 3.3.2 page 112. La construction des interfaces y est simplifiée et une multitude de comportements génériques sont disponibles pour construire les interactions. De plus, il est possible d'écrire nos propres interactions grâce au SDK (*Software Develoment Kit*) fourni pour développer nos propres boîtes comportementales. Nous avons déjà commencé l'étude de Virtools Dev et construit un premier prototype. Cet effort doit être poursuivi en proposant une interface permettant d'accéder aux deux activités de recherche et de lecture.

Nous devons également travailler sur l'interface d'annotation des documents afin de fournir un environnement complet. Avec le CNUM 3D, cet outil d'annotation pourrait être un simple traitement de texte. Un environnement mêlant 2D et 3D, comme le bureau 3D que nous présentons dans la section suivante, pourrait parfaitement convenir pour mettre en commun les différents outils. Il permettrait également d'exécuter un navigateur HTML pour obtenir des informations complémentaires sur un ouvrage.

La figure 4.28 illustre une telle interface. Dans celle-ci, sont représentés non seulement les éléments de la bibliothèque numérique mais aussi des applications 2D standards pour annoter les livres ou rechercher des informations complémentaires. Par ailleurs, on voit que la collection, dans cette figure, est schématisée sous la forme d'un arbre conique. Cela illustre un autre aspect que nous voudrions étudier : la souplesse de représentation de la collection. En fonction des préférences de l'utilisateur nous pourrions représenter la

collection dans un cylindre, une sphère, dans le lieu réel reconstruit ou encore un arbre conique, etc. Par défaut, jusqu'à ce qu'il marque sa préférence, on présenterait à l'utilisateur un modèle prédéfini par les bibliothécaires, qui sont les plus à même de savoir quelles représentations s'adaptent le mieux à leur bibliothèque. Un tri par défaut est également proposé pour cette même raison.

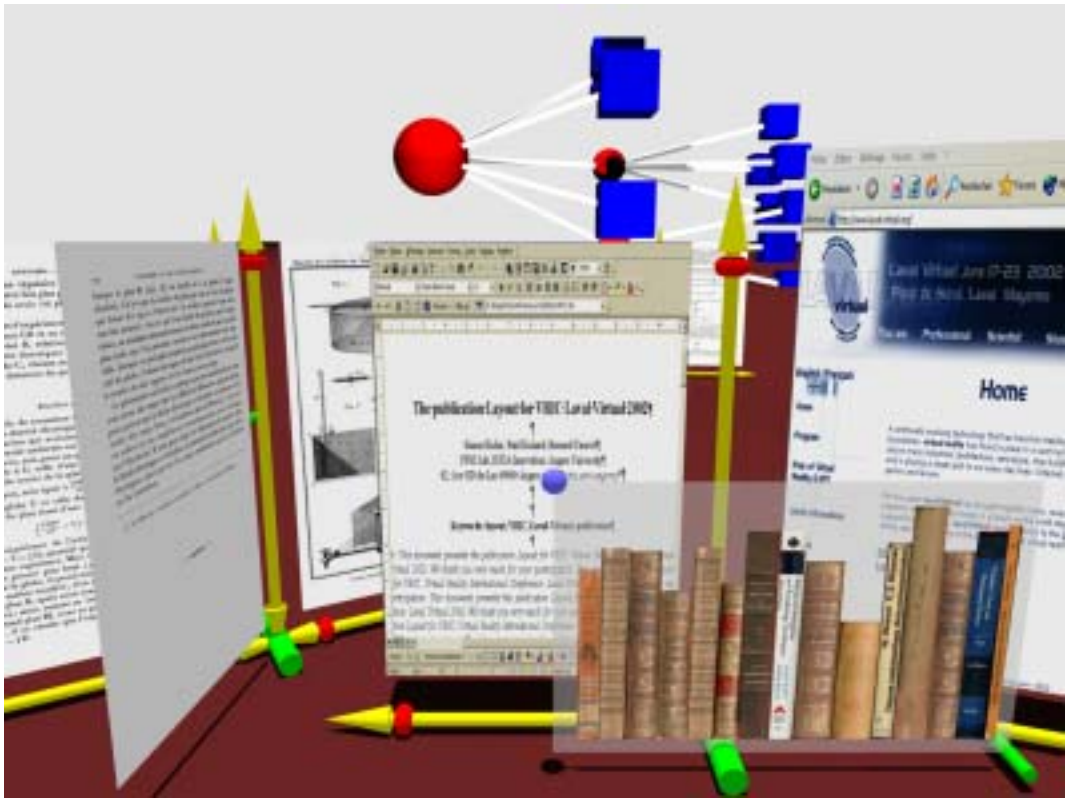


**Fig. 4.28** Applications 2D dans notre bibliothèque 3D.

## 4.5 Bureau 3D

L'essoufflement certain des techniques graphiques 2D pour représenter et visualiser les nombreuses données qu'un système multimédia peut stocker, est un constat reconnu. La tendance, avec les applications actuelles, est de réduire encore davantage la surface de travail, par l'ajout de barres d'outils et de barres d'état. La métaphore du bureau est tellement respectée qu'on peut facilement recréer le fouillis de documents visible sur un vrai bureau. Ce problème vient de la surface d'affichage très réduite disponible pour étaler les fenêtres recouvrantes (opaques). Comme nous l'avons vu dans la section 3.2 à partir de la page 63, de nombreux projets tentent, pour régler ce problème, d'élargir la surface (en 2D) ou le volume (en 3D) du bureau, l'endroit où sont placés et manipulés les documents et les applications.

Parmi ces techniques, les interfaces à changement d'échelle (section 3.2.2 page 67) utilisent les translations horizontales et zooms verticaux, ou bien des loupes ou autres outils pour modifier la taille des données. Ces interfaces font partie des interfaces 2.5D, englobant plus largement toutes les interfaces capables, soit de modifier l'échelle des données, soit de gérer les déplacements du point de vue dans le plan parallèle aux données (pas de rotations comme en 3D). Elles ont toutes démontrées l'importance des animations fluides pour la compréhension des modifications opérées. Elles ont également permis de mettre en évidence l'intérêt des espaces de travail, théoriquement infinis, pour organiser les données à visualiser.

Cependant, ces interfaces n'apportent rien de plus qu'une interface 3D. Nous les voyons plutôt comme une alternative temporaire aux interfaces 3D qui demandent une puissance de calcul assez importante. Or, la plupart des ordinateurs d'entrée de gamme vendus aujourd'hui disposent d'une carte graphique capable d'accélérer les opérations nécessaires à la 3D. Bientôt, les calculs nécessaires à la 3D représenteront le même pourcentage de temps d'utilisation du microprocesseur que les calculs requis par les interfaces 2.5D aujourd'hui. Ceci, parce que les microprocesseurs et les cartes graphiques 3D sont de plus en plus performants (voir la section 3.3 page 86). De plus, la 3D offre des possibilités supérieures à celles de la 2.5D comme l'utilisation des rotations et de la perspective. L'organisation des objets s'en trouve facilitée et les mouvements du point d'observation plus naturels. Pour élargir son champs visuel, en 3D, on tourne la "tête" (par rotation du point de vue) plutôt que d'effectuer, en 2.5D, une translation du corps entier.

### 4.5.1 Origine et besoins

Ce prototype de bureau 3D est né des expérimentations que nous avons faites sur les textures animées en OpenGL. Pour apprendre à gérer ces textures, nous avons utilisé les images de fenêtres du système XWindow dont le contenu était animé (par exemple, un mesureur de performance du système). Rapidement, il nous est apparu qu'il était possible d'intégrer toutes les fenêtres XWindow dans une scène 3D. Chaque fenêtre, dans la scène, étant représentée par une face texturée avec l'image de la fenêtre XWindow.

Le remplaçant du bureau 2D actuel n'a pas encore été trouvé. Cependant, nous sommes persuadés qu'il devra intégrer, à la fois, les applications avec une interface 2D et celles avec une interface 3D, qui sont de plus en plus nombreuses. Il n'est pas envisageable d'abandonner les applications 2D dont la plupart n'ont aucune raison d'être transformées en application 3D. De la même manière, les applications en mode texte ont été intégrées dans toutes les interfaces graphiques 2D grâce à l'utilisation de consoles texte.

Un sur-ensemble 3D permettrait donc, non seulement, d'englober les applications 2D mais également les applications en mode texte, elles-mêmes englobées par les interfaces 2D. D'autre part, la situation actuelle concernant la gestion des applications 3D n'est pas satisfaisante. Les applications 3D sont en effet exécutées dans une fenêtre 2D et forcent l'utilisateur à changer constamment de métaphores de navigation, lorsque qu'il passe d'une application avec une interface 2D à une application en 3D. Ce problème est constamment rencontré lorsque l'on visualise des scènes VRML. Les figures suivantes illustrent la diffé-

rence entre une application 3D (figure 4.29) exécutée dans un système de fenêtrage 2D et des applications 2D accessibles depuis une scène 3D (figure 4.30).

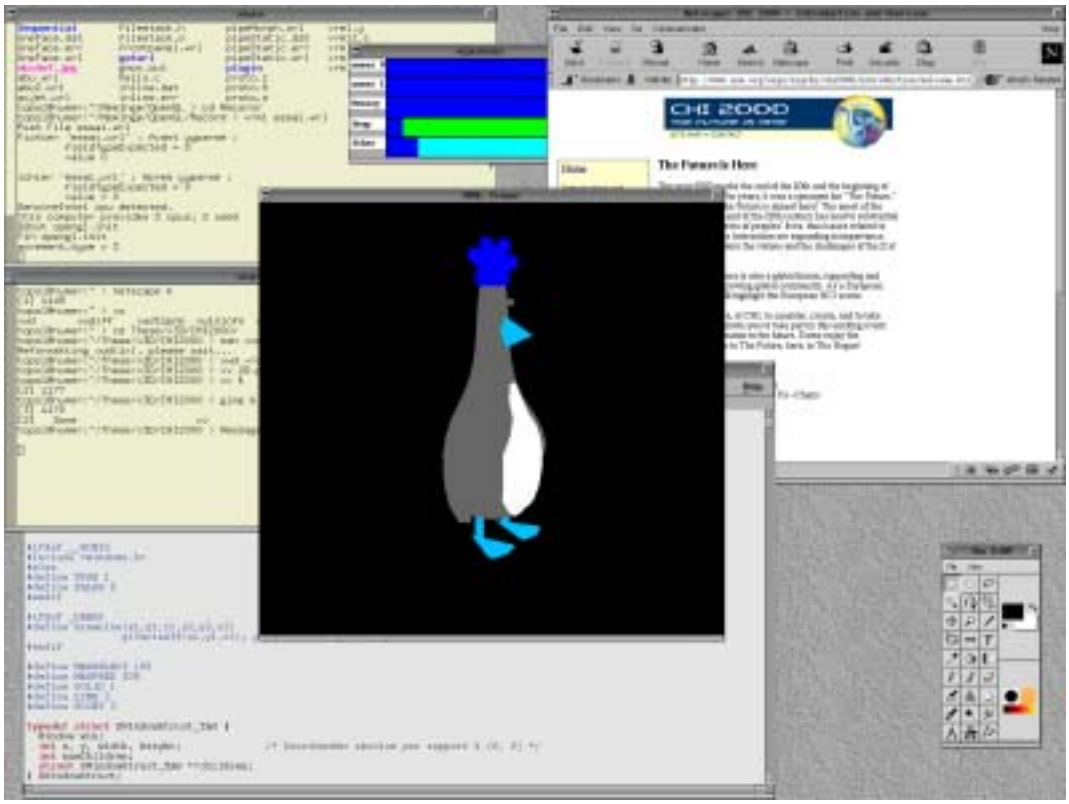


**Fig. 4.29** Une application 3D dans un gestionnaire de fenêtres 2D.

### 4.5.2 Les travaux proches

D'autres projets ont également eu pour tâche d'inclure des applications 2D dans un environnement 3D. Le gestionnaire de fenêtre MaW3D place les fenêtres dans un "tunnel" [Leach97] (voir page 56). Microsoft a également présenté un tel projet à la conférence CHI'2001, la *Task Gallery* [Robertson00] (voir page 58). La métaphore utilisée est celle de la galerie d'art. La ressemblance avec MaW3D est donc assez frappante puisque l'utilisateur est contraint à naviguer dans le couloir. Les fenêtres, dans ces deux tentatives, sont plaquées sur des murs. Nous pensons qu'une interface 3D ne doit pas être autant contrainte. Ces contraintes fortes n'autorisent pas l'inclusion d'objets 3D et rien (ni dans les articles, ni dans les copies d'écran) ne nous indique qu'il est possible de le faire dans ces deux environnements. D'autre part, l'apport des rotations, aussi bien du point de vue sur la scène que des objets, est considérable pour l'organisation d'un espace 3D. Cependant, cela nécessite d'étudier les politiques de placements et les techniques de manipulations.

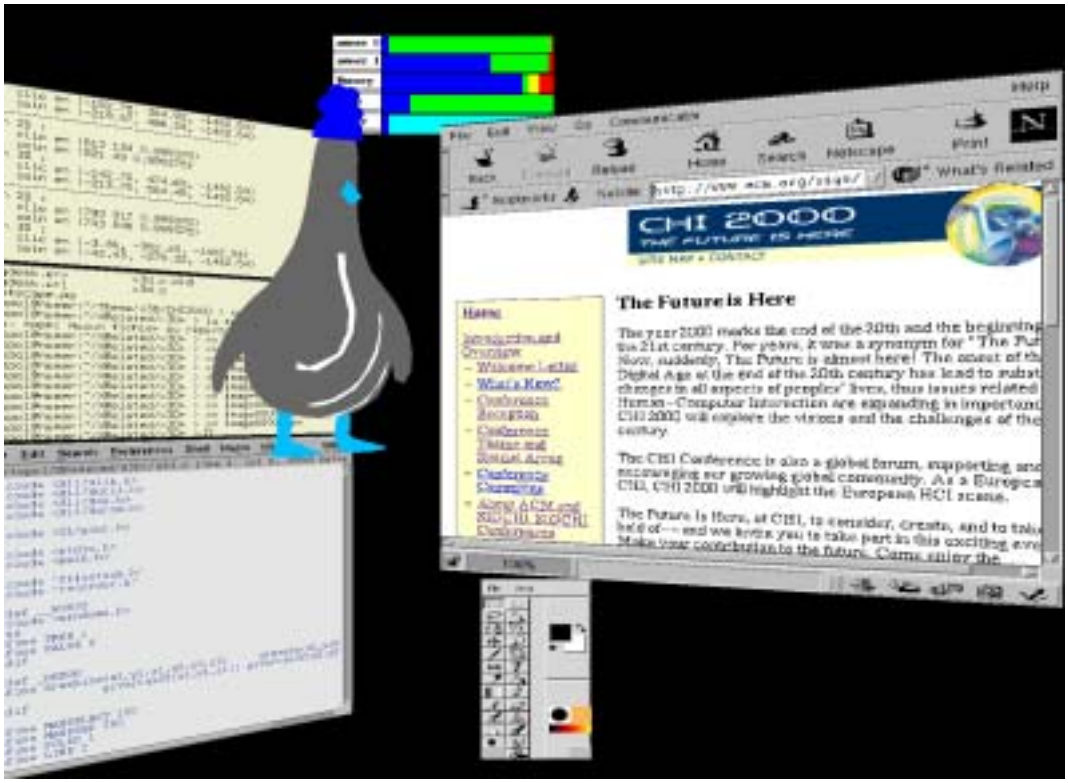


**Fig. 4.30** Les applications 2D dans un environnement 3D.

La deuxième tentative plus originale des chercheurs de chez Microsoft se propose de présenter les fenêtres sur un plan incliné. Dans la *Data Mountain*, les informations peuvent, en effet, être déplacées et positionnées où que ce soit sur ce plan incliné, par rapport au point de vue l'utilisateur (voir page 37). Les seules contraintes imposées sont que les documents reposent sur le plan et qu'ils ne sortent pas de ses limites. Dans cette interface, les fenêtres peuvent être positionnées en fonction de l'intérêt que leur porte l'utilisateur et plus encore, en fonction de sa propre notion d'importance du placement. Un utilisateur pourra placer les documents pertinents au premier plan parce qu'il peut ainsi les voir complètement à tout moment, alors qu'un autre les placera tout en haut du plan pour signifier leur plus grande importance. La mémoire spatiale sera donc complètement stimulée par cette notion personnelle d'importance attribuée aux informations. Cependant, pour ce projet également, l'utilisation d'objets 3D n'a pas été intégrée, en prévision des futures applications véritablement 3D.

Les projets les plus proches du nôtre sont sans conteste 3Dwm[9] et celui de N. Roussel [Roussel01] car ils ont été également développés sous Linux (voir page 57). Cependant, ces gestionnaires de fenêtres ont besoin, pour leur fonctionnement, de l'utilitaire VNC [Richardson98]. 3Dwm réorganise le *frame buffer* de XWindow sur une seule et unique face texturée avec l'image complète du bureau 2D de XWindow. Contrairement à notre prototype, chaque fenêtre n'est donc pas orientable et/ou positionnable indépendamment.

[9] A voir sur http ://www.3dwm.org.

Elles peuvent être déplacées mais doivent rester dans le plan réduit de la face. Cette solution, malgré une esthétique très travaillée, n'apporte donc rien de plus que la 2D pour la gestion des fenêtres. Elle offre néanmoins la possibilité de manipuler dans un même environnement aussi bien des objets 2D que 3D.

### 4.5.3 Description de notre projet

Notre gestionnaire 3D a été développé sur XWindow, car son architecture client/serveur permet de récupérer et de détourner l'usage des structures de données stockées par le serveur X. La même opération sous Windows nécessiterait de modifier le code de l'interface graphique (ce qui a d'ailleurs été fait par Microsoft pour implémenter la *Task Gallery*). Or, nous ne disposons pas de ces codes sources.

La première étape fut de récupérer les images des différentes fenêtres et de les inclure sur des faces OpenGL que nous avons ajoutées dans notre visualisateur de fichiers VRML. La récupération des images n'est pas un problème majeur. Nous avons implémenté exactement la même méthode que celle utilisée par les programmes de capture d'écran sous XWindow (comme xwd). Cette méthode est également utilisée par le logiciel VNC (*Virtual Network Computing*) d'AT&T, l'émulateur d'écran inter-plateformes.

La figure 4.31 schématise les différentes opérations effectuées lors de l'initialisation de la scène et lors de l'interaction sur une fenêtre. Pour l'initialisation, l'arbre des fenêtres est récupéré par un appel à la fonction `XQueryTree`. Pour chacune des fenêtres filles de la fenêtre *root* (principale), nous récupérons ensuite sa taille. La fenêtre est ensuite redimensionnée de telle manière qu'elle ait une taille de 2n * 2m, la taille de texture imposée sous OpenGL. L'image est ensuite récupérée et plaquée sur une face dont la taille est proportionnelle à l'image. Le sous-arbre des fenêtres de chacune des fenêtres de premier niveau, est ensuite sauvegardé pour déterminer, durant la phase d'interaction, sur quelle fenêtre l'utilisateur a agit. Il faut en effet savoir que sous XWindow, tous les éléments graphiques (*widgets*) sont des fenêtres.

Lorsque l'utilisateur effectue un clic de souris, on détermine d'abord sur quel objet il a eu lieu. Si c'est une fenêtre, les coordonnées 3D du point cliqué sont calculées, pour pouvoir déterminer les coordonnées 2D de l'endroit cliqué dans la fenêtre, par rapport à son coin supérieur gauche. Le *widget* sur lequel a eu lieu ce clic de souris est ensuite extrait grâce à la sauvegarde de l'arbre des sous-fenêtres. Les événements "bouton appuyé" et "bouton relâché" sont envoyés à ce *widget*. Les nouveaux *widgets* que cette action a pu créer sont stockés dans l'arbre des sous-fenêtres. Les modifications d'apparence de la fenêtre à la suite à cette interaction sont reflétées automatiquement par le moteur OpenGL lorsqu'il est inoccupé. Toutes les fenêtres sont donc périodiquement remises à jour par le moteur. Ainsi, les animations automatiques sont également visibles.

### 4.5.4 Développements futurs

Ce prototype répond à la même démarche que notre poste de lecture présenté précédemment : l'intégration des données 2D et 3D dans un même environnement, où les interactions

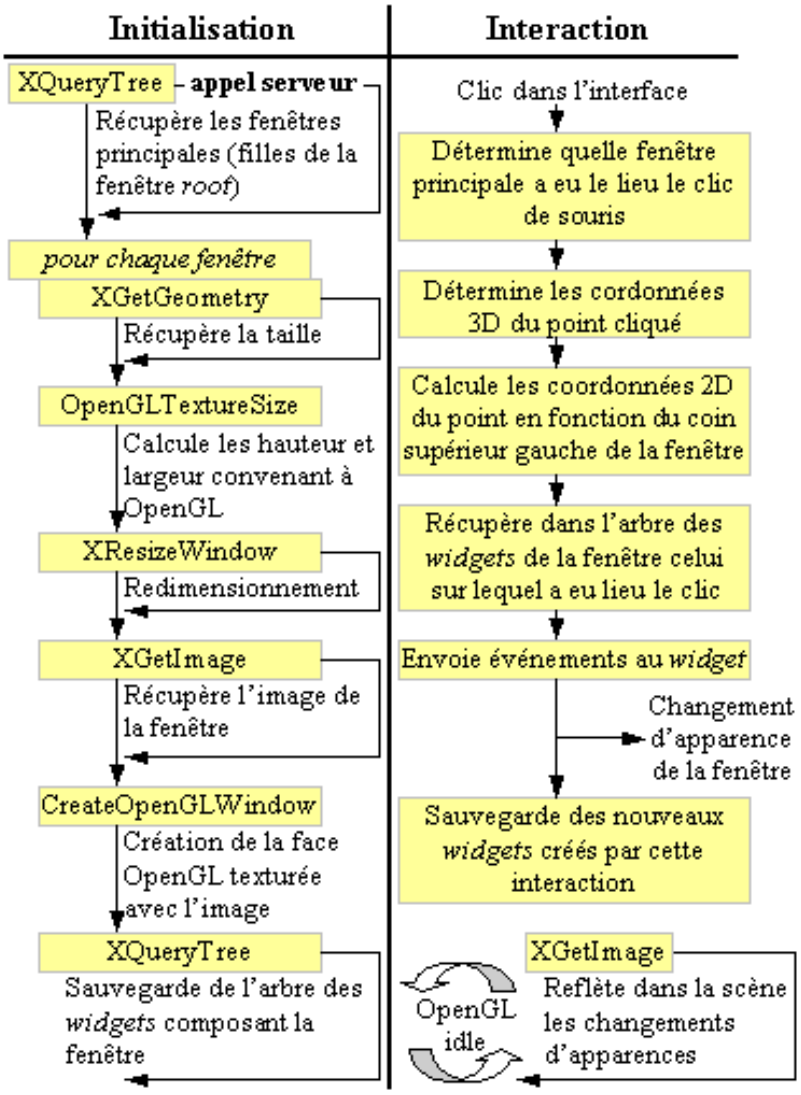


**Fig. 4.31** Initialisation et gestion des interactions avec le contenu d'une fenêtre 3D.

sont similaires sur les deux types de données. Cette solution peut être utilisée pour créer un poste de lecture puisqu'il est possible d'effectuer le rendu de scène VRML dans notre gestionnaire de fenêtre 3D (par l'intermédiaire de notre navigateur VRML écrit en C). Dans ce cas, tout éditeur de texte pourra servir d'outil d'annotation à l'intérieur même de l'environnement 3D. C'est le cas dans la copie d'écran de la figure 4.28. De plus, l'utilisateur pourra, dans un même environnement, à la fois consulter notre bibliothèque numérique et obtenir d'autres ressources en passant par un navigateur HTML. Bien que nous disposions de tous les éléments nécessaires (poste de lecture 3D, navigateur VRML et "environnement applicatif" 3D) pour mettre en place un tel prototype, ce travail n'a pas encore été fait.

Cette application a été développée rapidement et un intérêt particulier devrait être apporté aux interactions sur les fenêtres elles-mêmes et non uniquement avec leur contenu. Cela suppose d'étudier quelles sont les interactions nécessaires. En ce qui concerne l'évolution de ce prototype, elle peut prendre deux directions différentes :
- vers un gestionnaire de fenêtre 3D utilisant le protocole X tel qu'il existe à l'heure actuelle (cela suppose que toutes les coordonnées 2D reçues du serveur, soient converties en coordonnées 3D),
- vers un serveur X 3D où les coordonnées des fenêtres sont directement gérées en 3D.

Quelle que soit la méthode choisie pour faire évoluer ce prototype, il sera nécessaire d'apporter une attention particulière au rendu des textes. Des techniques d'antialiasing devront être utilisées pour adoucir les contours des textes et des images. La prise en charge des autres événements devra également être étudiée. En particulier, les événements clavier qui permettent à l'utilisateur de saisir des données et donc ne pas être limité à la simple manipulation du pointeur. Ce travail ne pose pas de véritable problème puisqu'il suffit d'acheminer les événements vers le fenêtre sélectionnée.

## 4.6 Son spatialisé

Nous avons étudié les possibilités de spatialisation sonore de VRML lors d'une étude sur le format de fichier à employer pour décrire un environnement sonore virtuel pour un projet artistique. Ce projet, le "tapis persan" imaginé par Cécile Le Prado, doit proposer aux utilisateurs déambulant dans un lieu physique, des sons synthétisés, en fonction de l'endroit où ils se trouvent, et restitués dans un casque. Ce projet qui peut sembler à première vue relativement simple à mettre en œuvre, nécessite plusieurs études :
- sur le format de fichier à utiliser,
- sur la manière de calculer les sons perçus par les utilisateurs,
- sur la manière de restituer les sons aux utilisateurs,
- sur la manière de distribuer les calculs sur différentes machines en fonction de la géométrie du lieu et du nombre d'utilisateurs.

C'est pour la première étude que nous sommes intervenus. Nous avons tout de suite pensé au langage VRML pour décrire la scène sonore. Or, le langage VRML souffre de carences au niveau de la gestion sonore. Alors qu'un fichier VRML pourrait servir pour la description d'environnements sonores 3D, cela est rendu impossible par l'absence d'attributs sonores des objets et par la pauvreté de la description des sources sonores.

### 4.6.1 Choix de la méthode de rendu

Le rendu sonore d'un environnement 3D peut être fait de deux manières différentes : en respectant les propriétés des géométries au milieu desquelles l'utilisateur évolue, ou en permettant un rendu qui n'a rien à voir avec la scène. Ce dernier est possible grâce à quelques variables globales qui définissent les propriétés sonores de la scène entière. Cette méthode permet donc de simuler l'acoustique d'une cathédrale alors que la scène décrit une minuscule pièce.

Le type de rendu sonore qui respecte ou non le volume de la scène considérée, peut être traité de deux façons différentes comme nous l'avons vu dans le court état de l'art sur la spatialisation sonore page 124. La différence entre ces deux méthodes est très similaire à la différence qui existe entre les méthodes du lancé de rayons et de la 3D temps-réel pour le rendu visuel d'une scène 3D [Natkin98] :

- par le respect des lois physiques de réflexion et d'absorption des sons sur chacun des objets,
- par la description d'un "matériel sonore" attaché à chaque objet donnant son comportement simplifié et sa manière de "réagir" aux sons qui le percutent.

Cette deuxième méthode permettrait de spécifier le pourcentage de porosité d'un objet au lieu de donner ses attributs physiques. Il ne sera pas nécessaire, comme dans la première de ces deux méthodes, de gérer les réflexions sur les objets voisins, ni de connaître les angles d'arrivée des sons sur les faces. Ainsi, le temps nécessaire aux calculs de ce que l'utilisateur perçoit à une position donnée, pourront être fait en temps-réel, grâce aux variables globales de l'effet de salle.

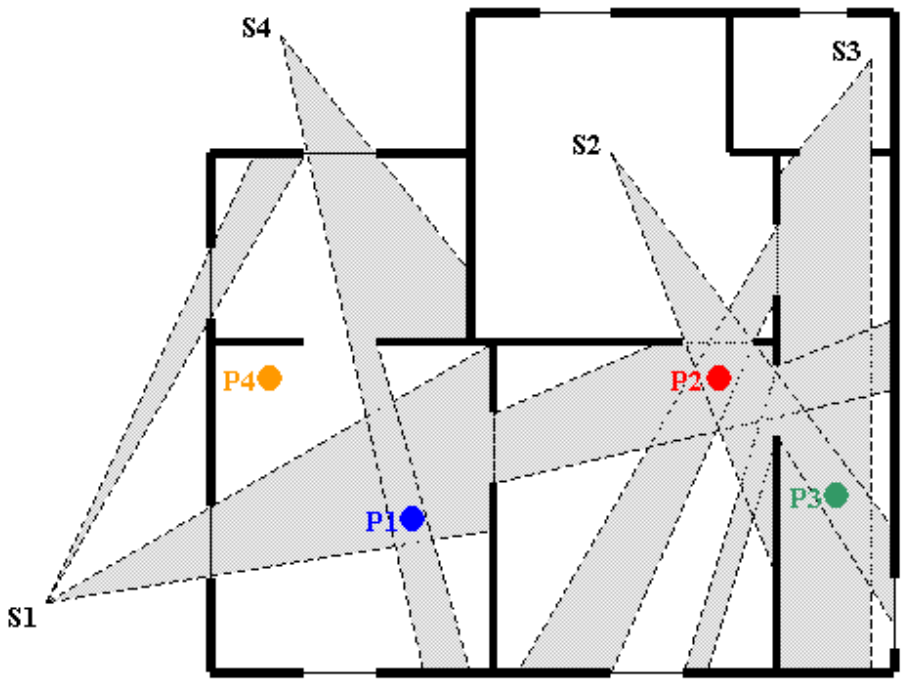


**Fig. 4.32** Exemple de scène virtuelle sonore.

Dans la figure 4.32, un exemple de ce que peut être une scène sonore est schématisé. Cette scène, vue du dessus, représente un appartement composé de différentes pièces. Les murs sont constitués d'un matériel sonore étanche, c'est-à-dire qui ne laisse pas passer les sons.

Les portes et les fenêtres, par contre, laissent filtrer les bruits. Dans cet environnement 3D, sont ajoutées quatre sources sonores ponctuelles. Pour chacune d'elles, en fonction de l'étanchéité ou non des surfaces qu'elle frappe, nous avons schématisé par des surfaces grises, les endroits dans l'appartement d'où cette source sonore peut être entendue. Le tableau suivant retranscrit, en fonction d'une des positions possibles de l'utilisateur schématisées dans la figure 4.32, les sources sonores qu'il entendra :

| **Position** | *Sources sonores audibles* |
|---|---|
| **P1** | S1, S4 |
| **P2** | S1, S2, S3 |
| **P3** | S2, S3 |
| **P4** | - |

Dans cet exemple, on aurait pu considérer que les murs étaient poreux aux sons. Cependant, cela aurait complexifié la figure. De la même manière, on aurait pu considérer que seulement certains murs (les murs porteurs par exemple) étaient étanches. Enfin, les portes et les fenêtres pourraient filtrer les sons, lorsqu'elles sont fermées et les laisser passer entièrement, lorsqu'elles sont ouvertes. Cette souplesse, ajoutée au filtrage d'un son en fonction de la distance séparant un point de la source sonore, permettent de spécifier des scènes sonores relativement réalistes.

On peut très bien imaginer d'autres comportements sonores pour les faces. En effet, le choix binaire (arrête/laisse passer les sons) n'est pas suffisant. L'unique possibilité de donner le pourcentage du volume du son qui peut traverser une face n'est pas non plus acceptable. Il faudrait des comportements plus souples, tels que la transformation simple des sons par des filtres prédéfinis ou que l'on décrirait.

### 4.6.2 Choix du langage

Pour le rendu sonore, la solution retenue a été d'utiliser la librairie OpenAL. La principale raison de ce choix, est sa ressemblance avec la librairie graphique OpenGL. Premièrement, OpenAL utilise une syntaxe proche de celle d'OpenGL : par exemple, pour le placement des sources sonores et de l'auditeur par des transformations. Deuxièmement, cette librairie sonore, comme la librairie graphique, est multi-plateformes et bénéficie de l'adjonction de librairies supplémentaires pour prendre en charge certaines fonctionnalités. En effet, alors qu'OpenGL peut être étendue par les librairies GLUT et GLU, OpenAL, de son côté, peut faire appel à des fonctionnalités d'EAX pour la gestion des effets de salle. De plus, OpenAL bénéficie de l'adjonction de la librairie ALUT (*OpenAL Utility Toolkit*), qui facilite la gestion des fichiers sons.

L'aspect le plus problématique était la détermination du format à choisir pour la description des scènes sonores. Plusieurs alternatives s'offraient à nous pour implémenter le projet artistique de Cécile Le Prado. Cependant, en y regardant de plus près, très peu de solutions semblaient acceptables pour la description des scènes. Nous les comparons très brièvement dans cette section.

Un premier tri a été fait en fonction de certains critères qui nous paraissaient essentiels. Ces critères, sous forme de questions, sont énoncés ci-dessous et ont permis de dégager deux concurrents. D'un côté la nouvelle norme MPEG4, et de l'autre, la norme VRML97 ou sa déclinaison sous forme de DTD[10] XML[11], X3D[12].

– Le modèle perceptif est-il ou peut-il être facilement intégré dans le format de fichier ?
– Le format de fichier est-il largement utilisé ? Est-il considéré comme une norme ? Y a-t-il des logiciels pour ces fichiers en charge ?
– Est-il facile de décrire une scène avec ce format de fichier ?

La figure 4.33 est fournie pour pouvoir peser les similitudes et différences entre ces deux formats de fichier, au niveau de la description des sources et matériaux sonores. Elle illustre le fait que toute la différence entre ces deux formats vient de la présence des AudioBIFS dans MPEG4 pour gérer le mixage et les effets avec plusieurs sources sonores. Bien que sous MPEG4, la description soit donc plus évoluée, nous avons néanmoins choisi d'utiliser VRML. Nous exposons les raisons de ce choix en détaillant les avantages et inconvénients des deux normes.

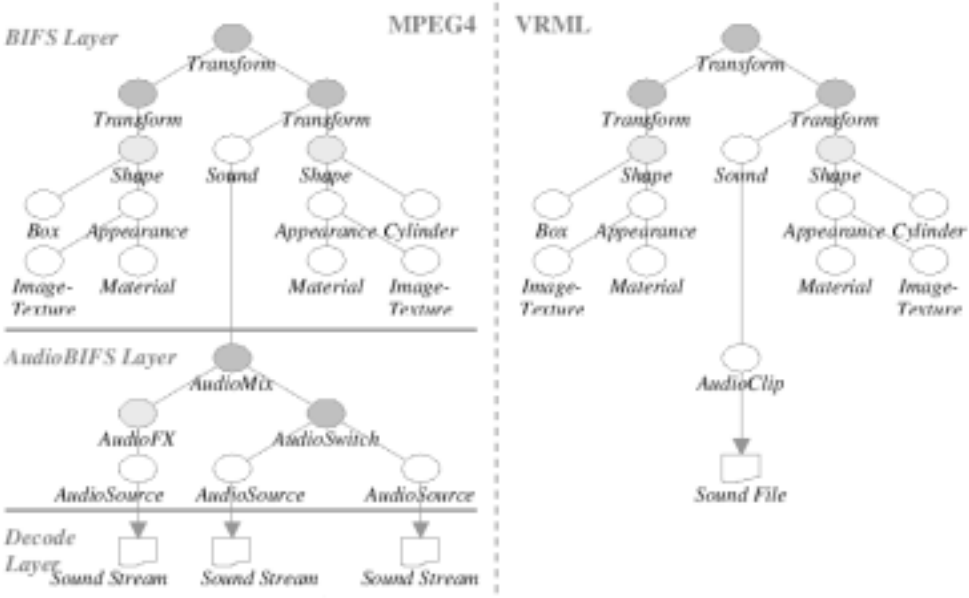


**Fig. 4.33** Les graphes de scène comprenant des nœuds sonores avec MPEG4 et VRML.

## MPEG4

MPEG4 permet, avec la deuxième version de ses audio BIFS, de gérer le rendu 3D sonore. Comme cela est expliqué dans [Scheirer99], les deux types de rendu sonore devraient être

[10] Document Type Definition.
[11] eXtensible Markup Language.
[12] Extensible 3D.

supportés. Le modèle physique est issu du projet DIVA [Huopaniemi96, Savioja97] développé à l'université d'Helsinki. L'approche perceptive (ou psycho-acoustique) est, quant à elle, inspirée directement par le spatialisateur développé à l'IRCAM [Jot95, Jot97].

Avec MPEG4, un graphe de scène sonore peut être attaché au graphe de scène principal par l'intermédiaire d'un nœud *Sound*. Comme avec VRML, ces nœuds sont soumis aux transformations préalablement définies dans les nœuds parents. La différence majeure vient de la manière dont est utilisé ce nœud *Sound*. Dans VRML, il ne représente qu'une source sonore spatialisée, c'est-à-dire positionnée dans l'espace. Avec MPEG4, ce nœud définit un sous-graphe de traitement du son. Comme on peut le voir sur la figure 4.33, des nœuds de traitement des flux sonores démultiplexés peuvent modifier le son avant qu'il ne soit spatialisé. Pour faire un rapprochement avec les API de spatialisation sonore 3D, MPEG4 peut être vu comme ayant les fonctionnalités d'EAX alors que VRML ne dispose que des possibilités de DirectSound.

Le processus complet de spatialisation avec MPEG4 est défini à l'aide de trois nouveaux nœuds :
- *AcousticScene* est un nœud de groupement définissant des propriétés communes à différents objets,
- *AcousticMaterial* permet d'attacher des propriétés sonores aux objets,
- *DirectiveSound* permet de définir un modèle de source sonore directive.

Le premier nœud correspond tout à fait aux variables globales nécessaires aux calculs des effets de salle. Le deuxième, quant à lui, définit des propriétés sonores simples pour les géométries. Enfin, le troisième, permet de décrire les sources sonores avec un modèle plus réaliste et souple que le modèle utilisé dans VRML, dans lequel des ellipses sont utilisées pour définir le volume à l'intérieur duquel le son est audible.

A la lecture de ses qualités pour le traitement du son spatialisé, on pourrait croire que MPEG4 satisfait parfaitement nos besoins. Cependant, aucune documentation n'est claire sur ses véritables possibilités. Il est même difficile de savoir s'il existe une application qui gère cet aspect. Nous proposons donc d'utiliser VRML comme langage support, car la description des scènes y est claire. De plus, l'ajout de nœuds et d'attributs est facile et puisque nous disposons d'un moteur de rendu visuel, nous pouvons expérimenter ces nouveaux éléments. Et dans la mesure où MPEG4 reprend les nœuds VRML pour la description des scènes, il sera relativement aisé de l'utiliser, lorsque la norme sera complètement disponible et compréhensible.

## VRML

Puisque rien n'est vraiment disponible pour prendre en charge les fichiers MPEG4 (ni format de fichier, ni ressources), nous avons donc fait le choix d'utiliser VRML. Lorsque ce projet avec S. Natkin et C. Le Prado [Natkin00] a débuté, il y a deux ans, la version 2 des AudioBIFS n'était pas publique. Sachant que cette version était pressentie depuis déjà deux ans, nous avions bon espoir de la voir arriver rapidement. Il n'en fut malheureusement rien, et nous avons donc utilisé VRML comme alternative. Nous pensons, néanmoins, que les rares informations que nous avons récoltées sur les AudioBIFS version 2 prouvent que

MPEG4 sera très puissant pour les traitements sonores 3D. C'est pourquoi, nous nous en sommes très largement inspirés.

En VRML, nous pourrions prendre en charge toutes les méthodes de rendu sonore précédentes. Mais le but de VRML est de permettre une interaction temps-réel. Pour ne pas l'éloigner de ce rôle, nous proposons donc de gérer directement les "matériaux sonores". Les sources sonores seront décrites de la même manière (nœud *Sound*) mais les "*mappings* sonores" des objets, permettront de mixer et filtrer les différents sons, de manière à obtenir une scène plus riche et acceptable du point de vue sonore. Les extensions que nous avons proposées à la suite de ce travail sont données dans la section 5.3.2 à partir de la page 212.

# Chapitre 5

# *Critiques et solutions proposées*

**Contenu du chapitre**

*Dans ce chapitre, nous présentons les enseignements tirés de nos expérimentations. Ils nous ont conduit à la proposition de certaines améliorations du langage VRML. Les nœuds comportementaux et sonores que nous ajoutons à la syntaxe de VRML sont décrits en détail.*

*Ce chapitre est organisé de la manière suivante :*


- Section 5.1 – Introduction du chapitre – p. 188 : *Traitement usuel des interactions et définition des comportements.*
- Section 5.2 – Enseignements tirés de nos expérimentations – p. 189 : *Constats généraux sur les différents projets.*
- Section 5.3 – Extensions du langage VRML – p. 199 : *Les ajouts que nous faisons à la syntaxe VRML pour gérer les comportements et la spatialisation sonore.*

## 5.1 Introduction du chapitre

Les expérimentations précédentes nous ont permis de mettre à jour quelques lacunes du langage VRML. Plus généralement, ce sont les langages permettant de créer une interface 3D, qui souffrent du manque de fonctions haut-niveau pour prendre en charge les interactions. Cette remarque est vraie pour la grande majorité des API 3D et des outils 3D (modeleurs, moteurs pour les jeux ou pour Internet). Une interface se compose nécessairement d'une partie en charge de la présentation des informations et d'une autre partie traitant les interactions. En 2D, toutes les facilités sont fournies aux programmeurs, via les IDE (les environnements de développement), pour simplifier les deux aspects. Il en résulte, qu'un programmeur a beaucoup moins de lignes de code à écrire.

Alors que de nombreuses fonctions complexes existent pour le rendu, aucune n'est disponible pour l'interaction en 3D. Or, il nous paraît que la 3D séduira plus les concepteurs et programmeurs d'applications, une fois qu'elle bénéficiera de fonctions facilitant le développement des interactions.

Dans notre cas, nous avons remarqué qu'une application moyennement complexe, décrite avec le langage VRML, se composait, pour plus des trois quarts, de scripts en Java. Nous reviendrons sur les problèmes que cela pose dans les sections suivantes. Pour tenter de régler ces problèmes, nous proposons d'intégrer quelques comportements génériques à la syntaxe VRML. Ils permettent, comme nous le verrons, de diminuer nettement la part de scripts nécessaires dans nos applications.

Quelles que soient les possibilités offertes par des comportements 3D, l'intégration d'éléments 2D reste nécessaire et importante à étudier. Comme nous l'avons déjà énoncé, d'une part, la 3D permet de régler en partie le problème d'occlusion des fenêtres "recouvrantes". Et d'autre part, il apparaît que le passage d'un paradigme d'interaction et de navigation à l'autre est gênant lorsque l'on bascule d'une application 3D à une application 2D ou vis-versa. Enfin, manipuler la caméra et les objets d'une interface 3D n'est pas une tâche aisée pour des utilisateurs novices. L'intégration de fenêtres, de menus, de boutons, ainsi qu'une manipulation optionnelle du point de vue, peut faciliter grandement l'apprentissage des trois dimensions.

Les comportements 2D des éléments présents dans toute interface WIMP sont connus. Néanmoins, leurs pendants pour une interface 3D doivent être étudiés. L'ajout de quatre degrés de liberté (la translation sur l'axe Z et les trois rotations) multiplie les comportements possibles. D'autre part, l'organisation de composants 2D ou 3D dans un objet conteneur doit également être considérée. Dans un premier temps, nous considérons l'ajout tel quel des comportements 2D dans un environnement 3D. Ils fournissent une première base sur laquelle il faudra néanmoins continuer de travailler. La seconde série de comportements que nous intégrons, sont les comportements nécessaires au fonctionnement de nos propres applications. Mais nous pensons que ces composants très basiques, sont utilisables par ailleurs.

Pour illustrer nos propos, nous utiliserons le langage VRML dans nos exemples et nos spécifications. Tout langage permettant de décrire un graphe de scènes aurait pu convenir mais nous préférons VRML car nous pensons que sa syntaxe est claire et que sa sémantique

est proche de celle utilisée par les bibliothèques de fonctions 3D qui manipulent les graphes de scène (OpenInventor, Java3D, Fahrenheit). De plus, c'est l'unique norme permettant de mettre en ligne une scène 3D décrite [VRML97]. Enfin, c'est le langage que nous avons principalement utilisé dans nos diverses expérimentations.

## 5.2 Enseignements tirés de nos expérimentations

Avant de parler des extensions que nous proposons, nous résumons tout d'abord les enseignements que nous ont apportés les expérimentations précédentes. Elles portent aussi bien sur les aspects ergonomiques ou conceptuels des applications que sur les aspects techniques.

### 5.2.1 Critiques des choix de conception

La plupart des choix liés à la conception ont, pour la plupart, déjà été discutés indépendamment pour chacune des interfaces 3D que nous avons présentées dans le chapitre 4. Dans cette section, nous mettons en commun les diverses critiques que nous avons faites et pouvons faire. Les plus négatives s'attaquent à la naïveté des choix de représentation, et en particulier, à l'utilisation d'une représentation en 2D des informations, pour faciliter leur maniement par les novices de la 3D.

Ces expérimentations nous ont tout de même permis de valider un aspect très important : le choix entre la 3D programmée et la 3D décrite. Nous revenons ici sur ce choix et sur nos arguments pour continuer notre étude dans ce sens, plutôt que de renoncer en développant une application sur mesure.

#### Naïveté des interfaces

Les conteneurs 2D mis en place dans deux de nos interfaces, le premier poste de lecture en VRML et le bureau 3D, s'avère être un choix très discutable. Le but recherché était de fournir aux utilisateurs des composants qu'ils ont l'habitude de manipuler. Pour les utilisateurs novices en 3D, en particulier, nous pensions que cela leur faciliterait la prise en main de l'interface. Les éléments 2D ont, certes, des fonctionnalités connues de tous, mais, ils ne s'adaptent pas bien à la 3D pour deux raisons principales.

La première raison est qu'ils n'utilisent pas de représentations 3D. Les seuls avantages de les immerger dans un environnement 3D est donc de profiter des rotations, de la transparences et du zoom interactif. Avec ces fonctionnalités, il est toutefois possible d'organiser autrement un espace de travail, et, pour cela, les études paraissaient intéressantes. Néanmoins, nous regrettons maintenant de n'avoir pas proposé une représentation 3D des éléments. La troisième dimension, comme nous l'avons vu avec le second prototype du poste de lecture développé avec Virtools, peut servir à rendre l'interface plus attractive mais surtout à intégrer de nombreuses fonctionnalités. La spatialisation des données n'en est pas gênée pour autant.

La deuxième raison se situe au niveau des interactions. Bien que les éléments 2D soient connus, et qu'ils bénéficient donc d'un apprentissage moindre pour savoir comment ils réagissent aux interactions, il faut toutefois appréhender leur comportements en 3D. On sait, en effet, comme fermer une fenêtre dans l'environnement 3D, mais, on ne sait pas comment la déplacer, l'orienter. Toute interaction ne passant pas par les *widgets* fournis (bouton de fermeture, d'agrandissment, d'icônification), doit être entièrement réapprise. L'apport des éléments 2D n'est donc pas, dans ce cas, intéressant. Ils n'aident pas les novices et limitent les experts en 3D.

Cependant, les interactions mises en œuvre pour les déplacements des fenêtres sont très simples. Elles ne subissent que des translations dans le plan (X, Y). La souris 2D ne se prête d'ailleurs pas bien à l'ajout d'autres interactions. On peut néanmoins orienter et déplacer librement les fenêtres en utilisant les métaphores de navigation présentes dans tout *browser* VRML. Cela nécessite de devoir sélectionner quelles fenêtres subissent les mouvements. Dans notre prototype du poste de lecture, nous avons implémenté une sélection inversée, en proposant de punaiser les fenêtres qui doivent rester immobiles. Il aurait sans doute été plus judicieux de fournir une opération de sélection normale.

Pour tenter de régler ces problèmes, nous avons proposé le deuxième prototype de poste de lecture. Il s'avère qu'un environnement entièrement 3D n'est pas plus difficile à utiliser qu'un environnement hybride 2D/3D. On accède, de cette manière, à la puissance représentative des interfaces 3D sans pour autant trop limiter les interactions.

### Intérêt pour la 3D décrite

Le deuxième enseignement que nous ont apporté ces expérimentations, a trait au choix entre 3D programmée et 3D décrite. Que ce soit au niveau du rendu ou de la prise en charge des interactions, la 3D décrite ne souffre d'aucune carence excessive par rapport à la 3D programmée. Nous considérons même qu'elle procure un certain nombre d'avantages.

Le premier de ces avantages est la possibilité de mettre facilement en ligne les applications décrites. Chose qui est rendue difficile pour les applications programmées. C'est toutefois possible en utilisant des applets Java3D, mais cette solution est très peu employée car elle nécessite l'installation des classes Java3D sur la machine cliente. Or, cette API de taille conséquente ne peut pas être récupérée par tous les internautes, et tous n'ont pas non plus les connaissances pour les installer. Pour les applications décrites, un *plugin* est nécessaire mais la plupart s'installent désormais automatiquement.

Le deuxième point important nous faisant préférer la 3D décrite à la 3D programmée est la facilité de partage des applications. En 3D programmée, il est relativement difficile de réutiliser des morceaux de codes sources car, souvent, ils effectuent des opérations spécifiques pour une application. S'agissant la plupart du temps d'applications "sur mesure", on y trouve moins d'intérêt dans le partage du code d'une application à l'autre. De plus, il est difficile de comprendre et d'utiliser un code écrit par une autre personne. Un code programmé contient en vrac, la définition des objets, les interactions, la gestion des erreurs, le chargement des fichiers, etc. Avec une application décrite, seuls les objets, voire certaines interactions, sont fournies. Il devient ainsi plus facile d'extraire les fragments de

code nécessaires. D'autant plus, si la syntaxe du fichier est claire, comme c'est le cas avec VRML.

VRML compte, par rapport à tous les autres langages ou systèmes pour décrire les applications 3D, d'autres avantages. Premièrement, c'est une norme, disposant donc d'une spécification claire. Deuxièmement, c'est un format de fichier exporté par la plupart des modeleurs professionnels tels que 3D Studio ou Maya. Mais il existe également de nombreux modeleurs spécialisés pour la mise en ligne de scène VRML, comme *Internet Space Builder*, *Beyond 3D*, etc. L'existence de tous ces outils ouvre VRML aux non-programmeurs. Grâce à eux, il n'est absolument pas nécessaire de connaître et comprendre la syntaxe VRML pour créer une scène et la mettre en ligne. Par ailleurs, grâce à l'exportation des scènes au format VRML, toutes sortes d'objets peuvent être convertis depuis les formats DXF, 3DS, etc.

Parce que c'est une norme, VRML a l'avantage de disposer de plusieurs *plugins*. Ces différents *plugins* comptent des atouts qui diffèrent de l'un à l'autre. Leurs interprétations de la norme VRML sont quasi identiques, mais ils se différencient sur les outils fournis pour naviguer dans les scènes, sur la rapidité du rendu, sur les extensions qu'ils intègrent à VRML. Ainsi, on pourra préférer *Cosmo Player* de SGI car c'est celui qui respecte la norme VRML au plus près, ou choisir *Cortona* de ParallelGraphics car il exploite plusieurs API pour le rendu et qu'il propose des extensions.

Pour les langages propriétaires, un seul *plugin* est généralement disponible. Il est, par conséquent, impossible d'en choisir un autre si des fonctionnalités déplaisent. De plus, le langage et le *plugin* peuvent évoluer complètement d'une version à une autre, sans que le propriétaire de la technologie se soucie de l'existant. On peut donc se retrouver du jour au lendemain avec des scènes non supportées. VRML étant une norme, il n'y a pas de risque de ne plus voir nos scènes supportées. Par exemple, la plupart des *plugins* VRML97 permettent encore la visualisation de scènes décrites avec le format VRML1.

VRML ne compte cependant pas que des avantages. La section suivante cite et illustre les problèmes que nous avons rencontrés pendant le développement de nos applications. C'est surtout dans le poste de lecture que sont apparus ces problèmes. Dans les autres projets, l'interface de recherche pour les bibliothèques numériques et le bureau 3D, VRML n'y est utilisé que comme langage de description des objets, et non comme langage pour décrire les applications 3D.

### 5.2.2 Insuffisances de VRML pour nos interactions

L'effort de programmation avec le modèle "bas niveau" de VRML (par l'adjonction de scripts en Java) n'est pas à la mesure des résultats obtenus. En aucun cas, un concepteur non programmeur ne pourra mettre à profit le système de scripts pour enrichir les interactions dans une scène VRML. Or, ce sont probablement les graphistes ou les artistes (donc des non programmeurs) qui sont en mesure de créer les scènes VRML les plus intéressantes. Nous pensons donc que l'utilisation des événements et des scripts est trop limitée, non pas pour ce qu'ils permettent de faire, mais plutôt pour le nombre de concepteurs potentiels à qui ils s'adressent.

De plus, pour les scènes "évoluées", les scripts représentent la plus grande partie du code diffusé. De ce fait, l'apport de VRML en tant que langage support est faible. Enfin, les scripts gérant de telles scènes sont difficilement distribués par leurs auteurs. Ils constituent en effet toute l'intelligence de la scène qu'un auteur ne veut pas toujours rendre visible pour de multiples raisons : ils sont leur savoir-faire, parce que leur code n'est pas propre ou mal commenté, etc. A ce jour, il est bien difficile de trouver sur Internet des scènes VRML scriptées utilisées à des fins non anecdotiques, ou de démonstration.

Le succès à large échelle de VRML repose sur une description des animations et des comportements aussi compréhensible que celle des objets. Cette simplification implique de réduire la place réservée aux scripts gérant les interactions. Nous introduisons pour cela des nœuds de comportements applicables à tout objet. Nous optons également pour le remplacement des routages par des mécanismes simples de contrôle des liens entre nœuds.

Nous différons sur ce point d'avec [Balaguer99], qui analyse fort bien les manques de VRML mais qui souhaite le faire évoluer vers un langage encore plus basé sur la programmation. C'est aussi le cas de l'architecture EMMA mêlant des objets 2D et 3D mais au prix d'une programmation complexe [Marrin98]. Avec notre solution, nous ne remettons pas en cause le modèle de description que nous trouvons puissant et proche des API 3D récentes (telles que Java3D, Direct3D, Fahrenheit). Cependant, pour pallier l'inexistence d'un modèle haut niveau pour décrire les comportements nous l'augmentons sans pour autant dénaturer sa structure.

Nous convenons néanmoins que le format de fichier VRML ne permet pas de décrire facilement des scènes interactives. Les quatre exemples suivants sont tirés de notre poste de lecture en VRML. Les interactions mises en œuvre sont très simples mais ne peuvent toutefois pas être décrites. C'est principalement pour décrire ces interactions et régler les problèmes suivants que nous avons intégré des comportements à VRML :
- les avantages de la description des scènes sont perdus par l'jout de code compilé,
- les scripts fonctionnent différemment (ou pas du tout) selon les *plugins* utilisés (pas de norme pour les scripts),
- les scripts sont très peu utilisés car ils ne sont pas sécurisés et révèlent le savoir-faire,
- les interactions ne peuvent pas être générées par un modeleur spécialisé,
- les scripts effectuent presque toujours les mêmes opérations.

Un langage VRML, augmenté de comportements les plus souvent utilisés, pourrait servir à décrire et mettre en ligne des applications plus facilement. Mais aussi, il pourrait être utilisé comme langage de prototypage d'applications compilées. Dans une première étape, il servirait à tester les interactions intégrées dans l'application, et, dans une seconde, à générer une application compilée optimisée, c'est-à-dire sans interactions générales superflus comme celles présentes dans les *plugins* 3D.

### Création/Destruction de contenu

Il est particulièrement important que VRML accède à la dynamicité pour étendre ses capacités. HTML a connu un sursaut formidable depuis qu'on été ajoutés les scripts en Javascript, les applets Java, ou encore les scripts CGI. Ces mêmes extensions sont possibles avec VRML au prix d'une programmation très lourde.

Dans notre poste de lecture, nous avons mis en place une gestion dynamique du graphe de scène VRML, à l'aide de scripts en Java. En fait, toute la scène est créée par les scripts : la collection récupérée sur le site CNUM, chaque fenêtre pour chacun des livres ouverts. Par exemple, à chaque fois que l'utilisateur clique sur un livre de la collection, l'application doit ouvrir une nouvelle fenêtre. De même, lorsque la souris passe sur un livre, une étiquette est créée, contenant le titre et le(s) auteur(s). Ces deux opérations nécessitent l'ajout et la suppression de nœuds dans le graphe VRML.

Pour cela, les scripts en Java doivent connaître l'état de la scène à tout moment. La figure 5.1 montre les ressources nécessaires pour créer les fenêtres. On y voit, en bas, la gestion des tableaux de fenêtres. Chaque case du tableau contient un identificateur de fenêtre unique et la position de cette fenêtre dans le groupe des fenêtres ; ces nombres ne sont pas nécessairement les mêmes après une série de suppressions et de créations.

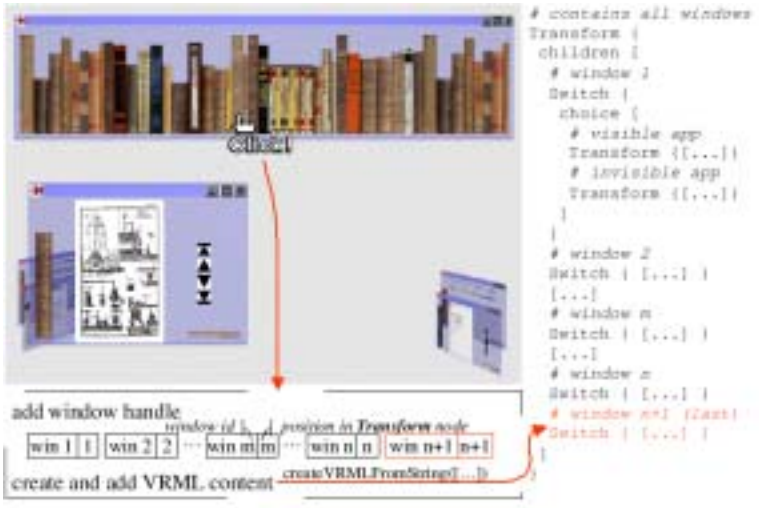


**Fig. 5.1** Création d'une fenêtre dans le poste de lecture en VRML.

A la droite de la copie d'écran, est schématisé le graphe de scène généré par les scripts au fur et à mesure que les fenêtres sont créées. La géométrie d'une fenêtre étant la même quel que soit le livre ouvert, l'ajout d'une nouvelle fenêtre correspond à un "copier-coller" dans le graphe de scène. Seul le contenu diffère en fonction du livre qui doit y être visualisé.

Le processus menant du clic de souris jusqu'à l'affichage de la fenêtre est simple mais la mise en œuvre est très lourde. Lorsque l'utilisateur clique sur un livre de la collection, un nouvel identifiant de fenêtre est créé. S'en suit la création du code VRML pour décrire la fenêtre. Ce code est inséré en dernière position dans le groupe des fenêtres et cette position est sauvegardée dans le tableau pour la retrouver ultérieurement. L'identifiant unique du livre cliqué permet enfin de décrire le contenu de la fenêtre dans une partie de son code VRML laissé vacant pour cela.

L'opération inverse est également nécessaire. Lorsque l'utilisateur n'a plus besoin d'une fenêtre, il la ferme. Le clic de souris sur le bouton "fermer" déclenche une série d'opérations menant à la suppression de la fenêtre à l'écran. Ces opérations schématisées dans la

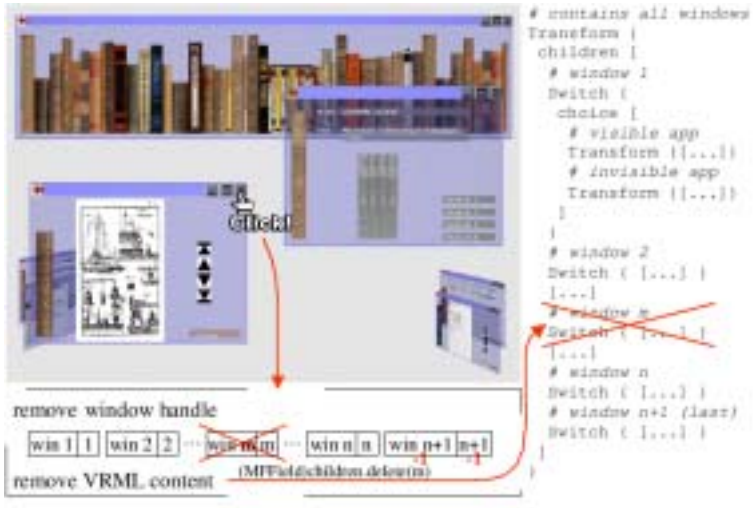


**Fig. 5.2** Suppression d'une fenêtre dans le poste de lecture en VRML.

figure 5.2, sont : l'identification de la fenêtre cliquée, la suppression de son code VRML correspondant et la suppression de son entrée dans le tableau. La fermeture est alors répercutée à l'écran lors de l'affichage de la trame suivante.

## Immobilité des objets

Une autre fonctionnalité présente dans notre poste de lecture, et nécessitant l'utilisation des scripts, est l'immobilité des fenêtres punaisées. Ces fenêtres ne doivent pas subir les mouvements de l'utilisateur. Plus exactement, leur immobilité est obtenue en leur appliquant la translation opposée de celle résultant du mouvement de l'utilisateur. La figure 5.3 montre les différentes étapes permettant de simuler l'immobilité d'une fenêtre punaisée.

Les opérations requises pour annuler l'effet de la navigation de l'utilisateur ne sont pas non plus très compliquées à comprendre. Mais, Elles nécessitent les scripts. On aurait pu n'utiliser qu'un capteur de proximité (*ProximitySensor*) et un routage d'événements vers un nœud de transformation de la fenêtre. Cependant, cette solution ne fonctionne que s'il n'y a qu'une seule fenêtre dans notre environnement. Le script sert en fait à déterminer quelles fenêtres doivent rester immobiles.

## Rangement des objets

Le principe précédent reste valable pour le rangement des fenêtres, illustré dans la figure 5.4. C'est-à-dire que les scripts sont nécessaires pour déterminer quelle fenêtre doit être animée et rangée. Une fois la fenêtre déterminée, on peut activer l'animation VRML spécifiée dans sa description. Les scripts sont également utiles dans ce cas pour calculer la position finale de la fenêtre pour qu'elle se range au bon emplacement.

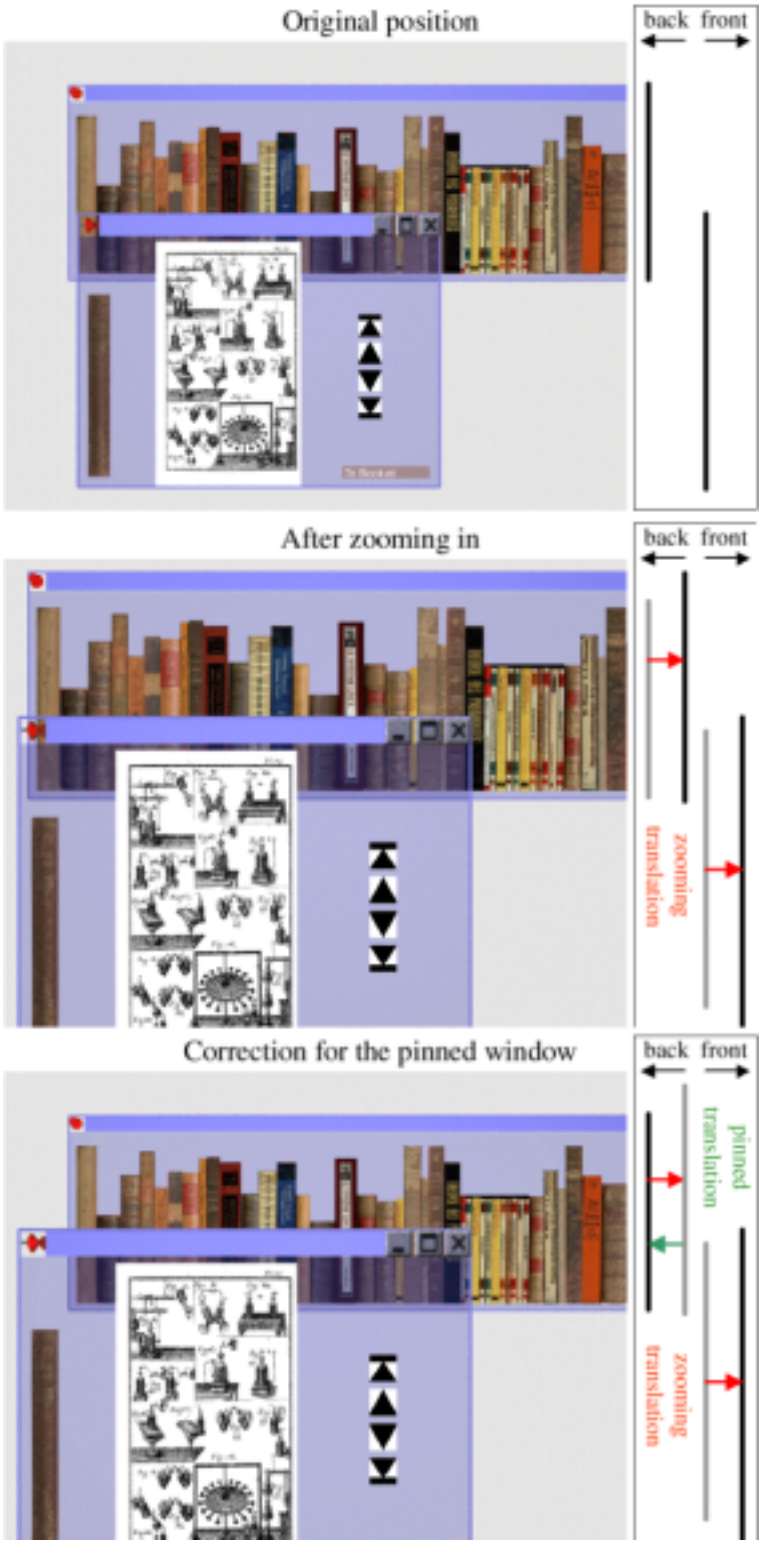


**Fig. 5.3** Immobilité d'une fenêtre punaisée dans le poste de lecture en VRML.

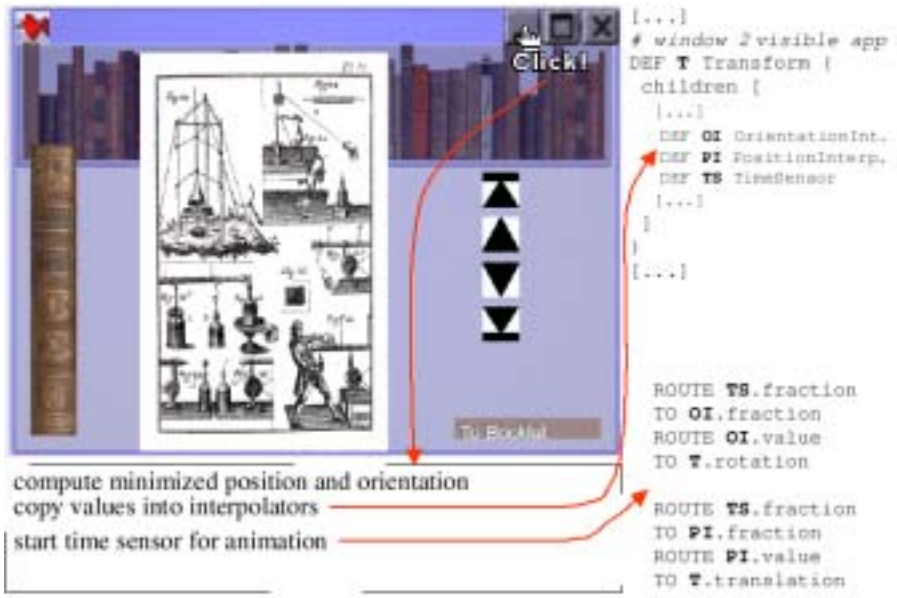


**Fig. 5.4** Rangement d'une fenêtre dans le poste de lecture en VRML.

La même stratégie est employée pour maximiser la fenêtre. Dans la figure 5.4, c'est une fenêtre maximisée qui est sur le point d'être rangée. Elle occupe tout l'espace d'affichage disponible. Cette maximisation nécessite de l'orienter perpendiculairement à l'axe Z, et de la positionner de telle manière qu'elle soit entièrement visible avec la plus grande taille possible. Les calculs prennent en compte, pour cela, l'angle de vue et la taille de la fenêtre de rendu. Une fois l'orientation et la position calculées, ces valeurs sont transmises à un nœuds *Interpolator* pour spécifier l'état d'arrivée de l'animation.

## Changement de textures

Le dernier aspect que nous décrivons pour mettre en évidence l'insuffisance de VRML pour décrire nos interactions, est le changement de page (figure 5.5).

Lors d'un clic sur un des boutons de navigation dans le livre, une requête HTTP est construite et envoyée, après ouverture d'une *socket* pour communiquer avec le serveur Web. La requête demande l'exécution d'un script CGI sur le serveur, qui renvoie l'URL de la page demandée. Cette URL est utilisée ensuite comme texture de la face représentant la page.

Le script adapte également la taille de la face représentant la page afin que ses dimensions soient proportionnelles à celles de la texture. Les nouveaux paramètres de la page, sa taille et sa texture, sont enfin insérés dans le code VRML décrivant le livre.

Ces différents exemples illustrent à quel point il est difficile de spécifier des interactions très simples avec VRML. C'est pour régler ces interactions que nous avons proposé d'utiliser des nœuds de comportement génériques (voir page 209).

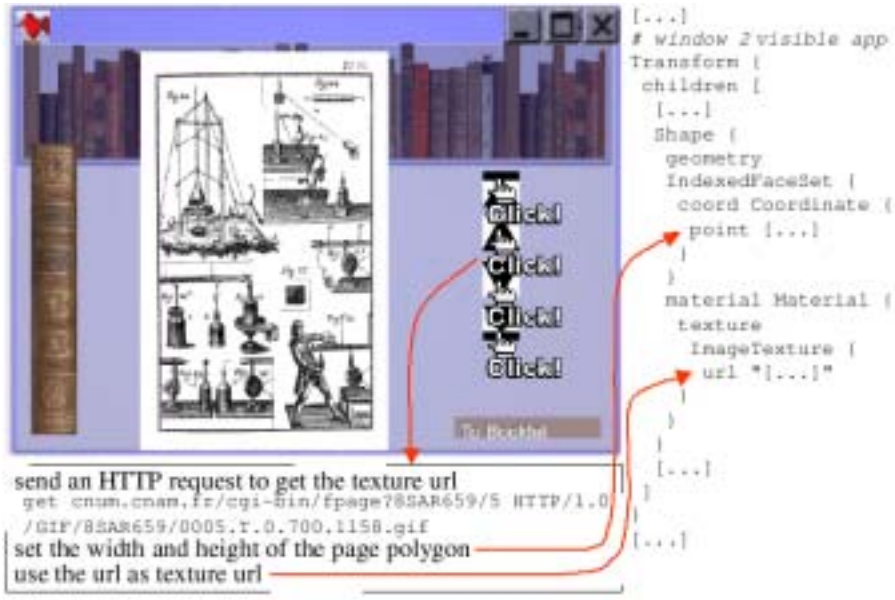


**Fig. 5.5** Changement de page dans le poste de lecture en VRML.

### 5.2.3 Insuffisances de VRML pour les scènes sonores

#### Ce que VRML peut faire

Le nœud *Sound* est utilisé pour attacher un son à un graphe de scène. Il spécifie également les propriétés sonores comme sa localisation, son orientation et deux ellipses qui définissent deux volumes à l'intérieur desquels le son est joué avec une intensité maximale et réduite progressivement jusqu'à extinction. Comme tout autre nœud VRML, les nœuds *Sound* subissent les transformations de leurs nœuds ancêtres. Cela signifie que les sources sonores peuvent être positionnées et orientées à l'aide des attributs *location* et *direction* du nœud *Sound*, mais elles peuvent également être déplacées et orientées par l'utilisation de nœuds *Transform*.

Les sons à jouer sont fournis par les nœuds *AudioClip* ou *MovieTexture* attachés au nœud *Sound* par l'intermédiaire de son attribut *source*. Ces nœuds fournissent des adresses où peuvent être récupérés les sons. Les formats de fichiers sons supportés, sont les formats MIDI et Wave. Dans le cas d'un nœud *MovieTexture*, la partie sonore d'une vidéo au format AVI ou MPEG sera extraite.

Dans sa version actuelle, VRML permet de positionner les sons dans l'espace et l'utilisateur les percevra différemment en fonctions de quatre paramètres. Les orientations du point de vue et de la source sonore sont utilisées pour calculer le son spatialisé que l'utilisateur perçoit. Les deux positions (de l'utilisateur et de la source sonore) sont utilisées pour calculer l'intensité du son. En fonction de la distance séparant l'utilisateur d'une source sonore et des deux ellipses définies pour la source, on peut rencontrer trois cas :

– lorsque l'utilisateur est à l'intérieur de la première ellipse, il entend le son avec son intensité maximale,
– lorsqu'il se trouve hors de la première ellipse mais à l'intérieur de la deuxième, l'intensité du son est calculée par une interpolation linéaire en fonction de la position de l'utilisateur par rapport aux frontières des ellipses,
– lorsque l'utilisateur est hors des deux ellipses, le son n'est pas perçu.

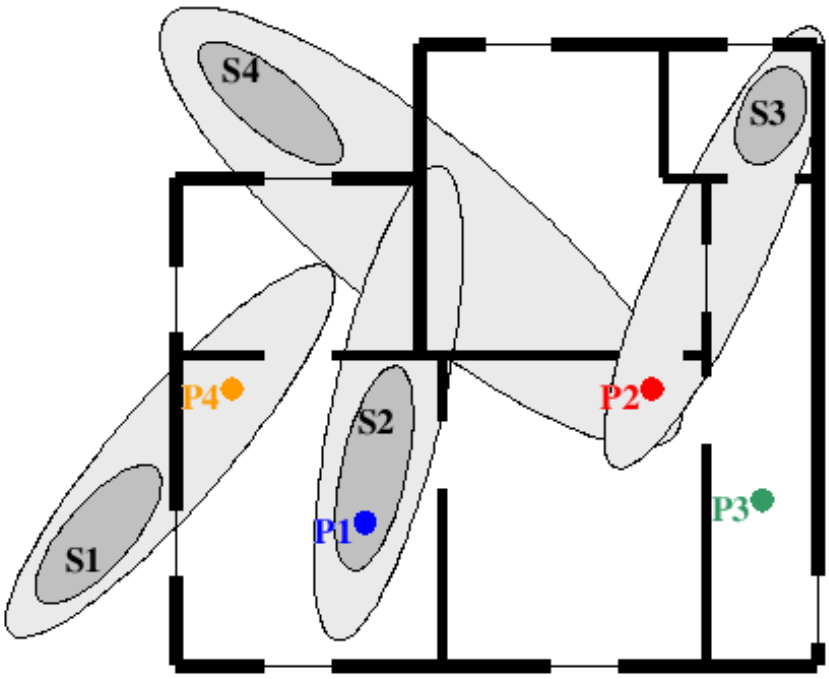


**Fig. 5.6** Exemple de scène sonore avec VRML.

La figure 5.6 illustre les différentes possibilités qui peuvent être rencontrées en fonction des paramètres des sources sonores et de la position de l'utilisateur. La scène utilisée est la même que dans la figure 4.32 page 181. Le gris foncé schématise les régions dans lesquelles les sons sont perçus avec leur intensité maximale. Le gris clair représente les zones dans lesquelles les sons sont perçus mais avec un volume réduit en fonction de la distance de l'utilisateur à la source. Le tableau suivante donne pour chacune des positions P1, P2, P3 et P4 schématisées dans la figure 5.6, quelles sources sonores S1, S2, S3 et S4 peuvent être perçues et avec quelles intensités approximatives.

| **Position** | *Sources sonores audibles* |
|---|---|
| **P1** | S2(100%) |
| **P2** | S3(20%), S4(10%) |
| **P3** | - |
| **P4** | S1(20%) |

On constate que les effets sonores sont très pauvres en VRML. Le premier, décrit ci-dessus, est un filtrage du volume en fonction de la distance séparant l'utilisateur et la source sonore (lorsque l'utilisateur se trouve dans la deuxième ellipses). Un second effet est le mixage de sources sonores. Par exemple, dans la position P2, l'utilisateur percevra un mélange de 20% du son S3 et 10% du son S4.

### Ce que VRML ne sait pas faire

Le résultat sonore obtenu par l'utilisation de zones elliptiques est très éloigné de ce qui peut être perçu en réalité. Par exemple, dans les positions P2 ou P4, l'utilisateur devrait entendre les sources S1 et S2. De plus, VRML ne permet de décrire les occlusions. A cause de cela, un utilisateur entendra un son avec la même intensité tant qu'il se trouve dans l'ellipse d'intensité maximum, même s'il est séparé de la source sonore par une géométrie. Dans certains cas, en fonction des ellipses définies, l'utilisateur peut percevoir une source sonore cachée derrière plusieurs géométries alors qu'il ne percevra pas une source sonore non cachée.

Les sons peuvent arrêter leur propagation après une distance donnée, mais dans toute les directions, et non uniquement dans un zone elliptique. Ces ellipses ne sont pas adaptées à la description sonore et elles sont difficiles à utiliser. Elles doivent être remplacées par une description plus réaliste et compréhensible. Des cônes sonores semblent mieux convenir car ils peuvent être définis de manière intuitive et leurs paramètres sont simples. De plus, avec une telle description des sons, un utilisateur entendra toujours un son se trouvant en face de lui, lorsqu'il n'y a pas d'obstacle entre les deux.

Ce qu'il est nécessaire d'inclure à VRML, est la gestion des réflexions et des réverbérations. Il faut également permettre aux objets présents dans la scène de modifier un son. Ces deux modifications contribueraient largement à rendre les scènes sonores en VRML beaucoup plus réalistes.

## 5.3 Extensions du langage VRML

Comme nous l'avons vu dans la section 3.3.3 page 114, dans la grande majorité des applications 3D, les interactions sont gérées par exécution d'un code écrit à l'aide d'un langage de programmation. Notre but ici, n'est pas de montrer que les comportements programmés associés à une scène décrite sont inutilisables. Au contraire, forts de notre expérience sur le poste de lecture, nous croyons que des scènes 3D complexes peuvent être créées de cette façon. Dans notre cas, l'utilisation de scripts en Java nous a permis d'effectuer des opérations non directement liées à la manipulation du graphe de scène en VRML comme la récupération des URL des pages des livres. La proposition d'un graphe de scène dans lequel tous les comportements standards sont décrits permettrait donc de réduire la part de code nécessaire. Cela les rendrait également accessibles aux non-programmeurs et aux générateurs de scènes. Cependant, elle ne supprimerait pas complètement les scripts dans tous les cas, puisqu'ils peuvent effectuer des opérations sur d'autres données que celles décrivant les objets graphiques (la récupération de l'URL d'une page par exemple).

Par ailleurs, nous présentons dans cette section, à partir de la page 212, les extensions sonores que nous avons intégrées à notre navigateur VRML.

### 5.3.1 Comportements

La solution d'adjoindre des comportements à la syntaxe de VRML est consistante puisque les comportements y seront décrits au même titre que la forme et l'apparence des objets. D'autre part, les bibliothèques de composants interactifs seraient distribuées plus facilement. A l'heure actuelle, il n'existe que peu de scènes hautement interactives sur Internet. Ce problème vient d'une protection des scripts gérant les comportements par leurs auteurs. En effet, il est relativement facile de dé-compiler un scripts Java pour en obtenir le code source. Or, ces scripts représentent tout le savoir présent dans une scène. Dans l'industrie, ce problème d'espionnage est réglé en utilisant un autre langage où les comportements ne peuvent pas être "violés". Un format de description règlerait ce problème s'il permettait de spécifier en très peu de lignes les comportements nécessaires. Et nous pensons alors que ces quelques lignes ne mériteraient plus d'être gardées jalousement secrètes.

Les comportements génériques et la description de comportements permettent de limiter la part du code requis. La connaissance d'un langage de programmation n'est alors plus nécessaire. Le développement des interactions s'en trouve facilité grâce à la suppression de l'étape de compilation usuellement nécessaire lorsque l'on utilise les scripts.

La figure 5.7 est un exemple de graphe de scène dans lequel sont introduits les comportements. Ils peuvent être insérés dans les nœuds de regroupement (c'est-à-dire les nœuds qui peuvent en contenir d'autres) par l'intermédiaire d'un type de nœud spécifiant une interaction ou une animation générique. Le principe et le fonctionnement de cette solution seront décrits plus en détail dans la section suivante. Le point important que nous voulons illustrer est l'importance de la dé-corrélation entre les objets intervenant dans le rendu et ceux servant à gérer les comportements. Cependant, ce problème devra être pris en charge au niveau de l'implémentation du gestionnaire des scènes et non au niveau de la description de celles-ci. Ici, les comportements sont directement présents dans le graphe de scène et donc *a priori* liés à la fréquence de rendu des objets graphiques. L'interpréteur de scène peut cependant créer une seconde structure, non nécessairement arborescente, qu'un moteur de comportements gérera pour en extraire les interactions et animations de manière indépendante. Java3D dont les codes de comportements sont intégrés au graphe de scène fonctionne sur ce principe.

Les mises à jour du moteur de comportements seront alors reflétées lors du rendu de la trame suivante. Certaines étapes visuelles d'une animation ou d'une interaction peuvent être et seront probablement occultées par le rendu. Mais l'utilisateur ne sera pas troublé par ce phénomène puisque la navigation dans la scène se fera avec la même fluidité. Il est préférable néanmoins d'avoir un taux de rafraîchissement suffisant pour obtenir des animations fluides et autoriser ainsi l'utilisateur à se reposer sur son système sensoriel pour percevoir les informations.

Si la fréquence du moteur de rendu est inférieure à celle du moteur de comportements, les animations des objets pourront apparaître comme des sauts plutôt que comme des

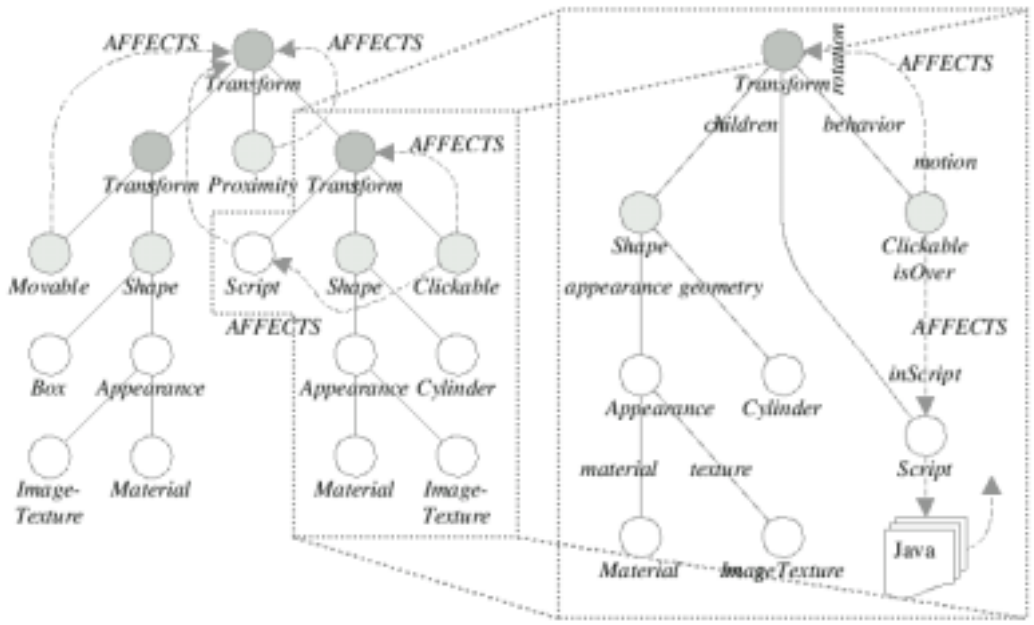


**Fig. 5.7** Graphe de scène VRML avec comportements.

transitions fluides. Le système perceptif de l'utilisateur sera alors mis à rude épreuve. Même si le rafraîchissement de la scène se fait à des fréquences rapides (en dessous du 25ème de seconde) mais toujours inférieures à celles du moteur de comportements, les sauts d'une position de l'animation à une autre se verront. Ce problème est en effet également dépendant de la durée de l'animation ainsi que de la longueur du trajet de l'objet animé. Pour une animation dont la trajectoire à une longueur totale D et une durée T on peut considérer qu'elle sera fluide si : $T.\Delta D/D < F$ avec $\Delta D$ la longueur des micro-mouvements acceptables et F la fréquence des trames successives calculées par le moteur de rendu. Si cette inéquation n'est pas vérifiée, dans le meilleur des cas l'animation sera rafraîchie à une fréquence de $2.T.\Delta D/D$ comme on peut le voir dans la figure 5.8.

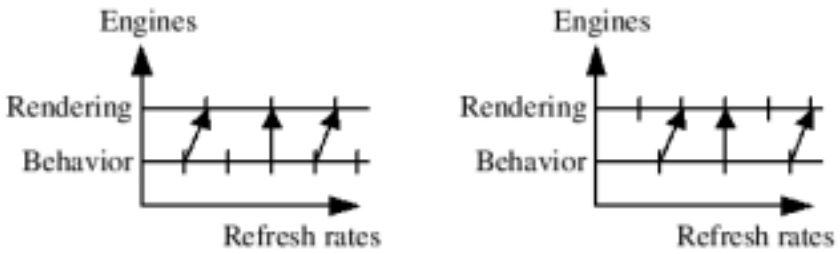


**Fig. 5.8** Taux de rafraichissement des moteurs de rendu et d'animations.

Chaque objet, composé ou non de plusieurs primitives graphiques, est alors caractérisé par son comportement propre mais également indirectement par le comportement de ses composants. Cela permet de propager les interactions de l'élément le plus élémentaire vers le plus complexe. Cela nous rappelle le fonctionnement des *widgets* [Flanagan94] [Nye93]. Le bloc en détail dans la figure 5.7 illustre bien le fait qu'un objet dispose des mêmes propriétés qu'un *widget* :
- une forme et une apparence,
- des comportements privés, définis par les comportements de ses sous-composants, qui n'affectent que l'objet lui-même,
- des comportements publics qui affectent d'autres objets ou des scripts assimilables à des fonctions de type *callback*.

## Comportements Vs Composants

Bien que la cascade de comportements et la propagation d'événements nous rappelle le fonctionnement des composants graphiques (*widgets*), il existe cependant quelques différences notables entre ceux-ci et les comportements que nous voulons mettre en œuvre. Nous expliquons dans cette section les diférences entre les composants standards que nous rencontrons en 2D et les comportements autonomes que nous proposons. Cette autonomie des comportements vis-à-vis de l'aspect graphique des objets peut être appliquée à la 2D.

Notre but est de pouvoir décrire les objets 3D comme des *widgets*, mais en séparant l'aspect graphique de l'aspect interaction. Autrement dit, nous ne proposons pas des *widgets* élémentaires mais des comportements élémentaires qui permettent de construire toutes sortes de *widgets* 3D.

Le résultat visuel obtenu pour un objet est le même. Cependant, les comportements peuvent être utilisés sur n'importe quel objet. Les *widgets* 2D élémentaires peuvent être considérés comme des boites noires définissant une forme, une apparence et des comportements que l'on ne peut pas modifier. Avec une spécification des comportements détachée de l'aspect visuel, on peut définir facilement, par exemple, un objet ayant les comportements d'un bouton sans pour autant qu'il en ait l'apparence usuelle.

La figure 5.9 schématise deux composants presque semblables. Il s'agit de deux boutons ayant une apparence différente mais des méthodes semblables. Les méthodes internes et externes sont respectivement les interactions intervenant sur le composant et celles déclenchant l'exécution d'un code externe au composant. Bien que leurs méthodes internes et externes soient les mêmes, deux composants doivent cependant être créés. On a en fait autant de composants à créer qu'il n'y a d'ensembles {apparence + comportement} différents.

La figure 5.10 propose, quant à elle, l'utilisation d'un comportement "bouton". Ce comportement pourrait être tout simplement une légère translation en profondeur de la géométrie lorsque l'utilisateur clique dessus. Il peut être adjoint à une multitude de géométries sans se soucier de leur apparence. Il n'y aura, dans ce cas, qu'à créer un seul comportement pour obtenir toutes les apparences de boutons fonctionnant de la même manière.

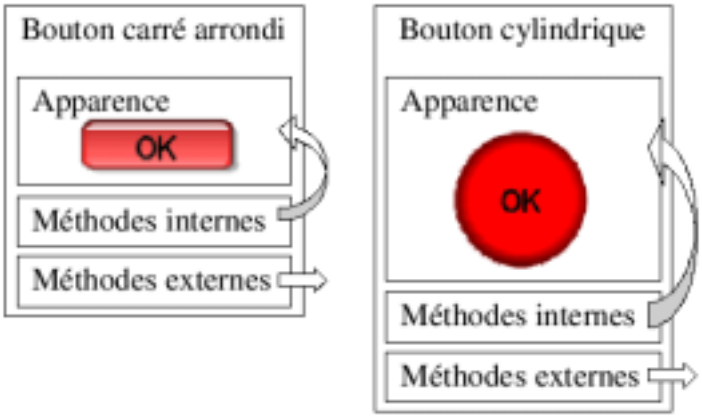


**Fig. 5.9** Exemple de composants différant par leur apparence.

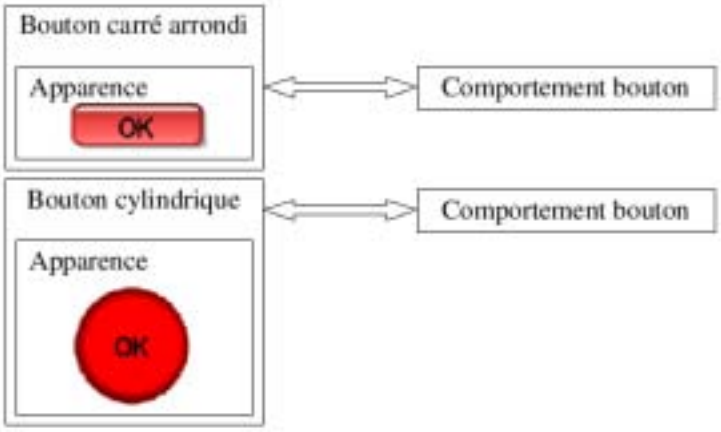


**Fig. 5.10** Exemple de composants différant par leur apparence.

On peut définir un comportement comme une série d'entrées qui, lorsqu'elles sont stimulées par un événement, effectuent certaines opérations, et fournissent un résultat en sortie. Dans la figure 5.11, nous avons schématisé le comportement permettant de ranger une fenêtre de notre poste de lecture. L'interaction y est décrite comme l'exécution d'un code associé à un événement. Le comportement d'un objet peut être défini comme l'ensemble des interactions auxquelles répond l'objet. Dans l'exemple, il est plus précisément sensible aux clics de souris et émet en sortie la position et l'orientation de rangement de la fenêtre. Ces informations sont affectées à un nœud d'interpolation pour mettre en place l'animation. Ce comportement pourra être réutilisé pour ranger n'importe quel objet 3D. Il suffira de l'intégrer dans le graphe de scène VRML et d'effectuer les branchements nécessaires avec les objets.

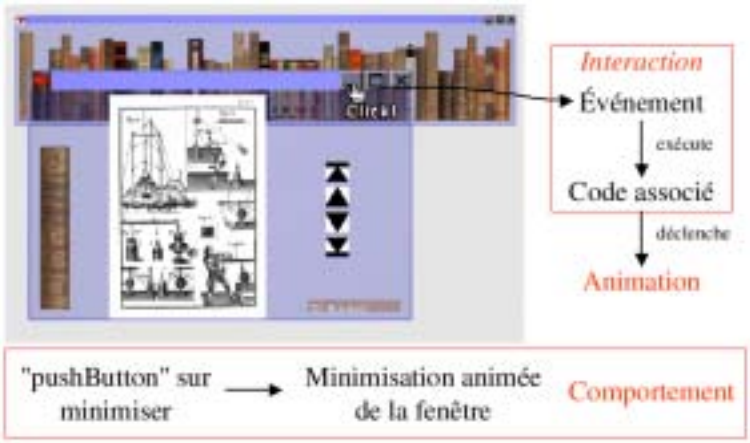


**Fig. 5.11** Mise en rapport des termes "comportement", "interaction" et "animation".

## Extraction de comportements

Nous avons fait le choix d'utiliser des comportements génériques. Une autre solution aurait pu être l'utilisation de comportements physiques. Mais, ceux-ci sont complexes et difficiles à décrire. De plus, il peuvent convenir à la création d'un simulateur ou d'un jeu, mais absolument pas à la création d'une interface 3D.

Les premiers comportements que nous avons extraits sont issus des prototypes que nous avons développés. Nous nous sommes en effet aperçus que les mêmes comportements étaient souvent utilisés, et que, par conséquent, nos applications futures bénéficieraient de la spécification de telles interactions. Ils proviennent essentiellement de deux sortes d'interactions différentes :

- les actions opérées par l'intermédiaire des dispositifs de pointage (plus particulièrement la souris 2D) : sélection, *roll-over*, *drag & drop*, etc.
- les politiques de placement 2D et 3D.

Les différents comportements extraits peuvent être agencés dans un graphe par insertion de nœuds gérant l'ordre des opérations. Cette fonctionnalité est particulièrement utile lorsque, à la suite d'une interaction, plusieurs opérations doivent être effectuées sur un ensemble de nœuds. Les nœuds d'ordonnancement (*SequentialAnim* et *ParallelAnim*) sont uniquement des nœuds contenant les opérations à exécuter. Lorsque les nœuds ne doivent subir qu'une seule modification, le nœud de comportement peut être directement intégré dans le graphe de scène, comme dans l'exemple suivant :

```
Clickable {
  children [
    DEF monObjet Transform {
      children [
        Shape { [...] } # Les noeuds constituant un objet quelconque
        [...]
        Shape { [...] }
```

```
      }
    }
  ]
  buttonPressed AFFECTS monTimer.start
}

DEF monTimer TimeSensor {
  [...]
  loop TRUE
  fraction_changed AFFECTS maRotation.set_value
}

DEF maRotation RotationInterpolator {
  key [...]
  keyValue [...]
  value_changed AFFECTS monObjet.set_rotation
}
```

Cet exemple est un extrait d'une scène permettant de déclencher une animation simple, en l'occurrence une rotation. Le nœud *Clickable* a ici la même utilité qu'un nœud *TouchSensor*. Néanmoins, nous préférons le premier, car il englobe les géométries sensibles aux clics de souris. Le nœud *TouchSensor* active, quant à lui, les nœuds se trouvant au même niveau dans le graphe de scène.

La figure 5.12 schématise un graphe de comportements. L'ordonnanceur exécute en parallèle ou en série les comportements intégrés dans le graphe de comportements. Les comportements qui peuvent y être inclus sont les comportements génériques ou des comportements décrits par l'utilisateur. Les comportements inclus dans ce graphe agissent sur les géométries (sur le graphe de scène VRML) à l'aide du mot-clé *AFFECTS* qui remplace le routage habituel de VRML. Nous reviendrons sur ce point dans la partie débutant à la page 209.

L'exemple suivant illustre la spécification d'un ordonnancement de différentes tâches. Il étend simplement l'exemple précédent en y ajoutant une translation à effectuer en parallèle avec la première rotation, et une seconde rotation à déclencher à l'issue des premières transformations.

```
Clickable {
  children [
    DEF monObjet Transform {
      children [
        Shape { [...] } # Les noeuds constituant un objet quelconque
        [...]
        Shape { [...] }
      }
    }
  ]
```

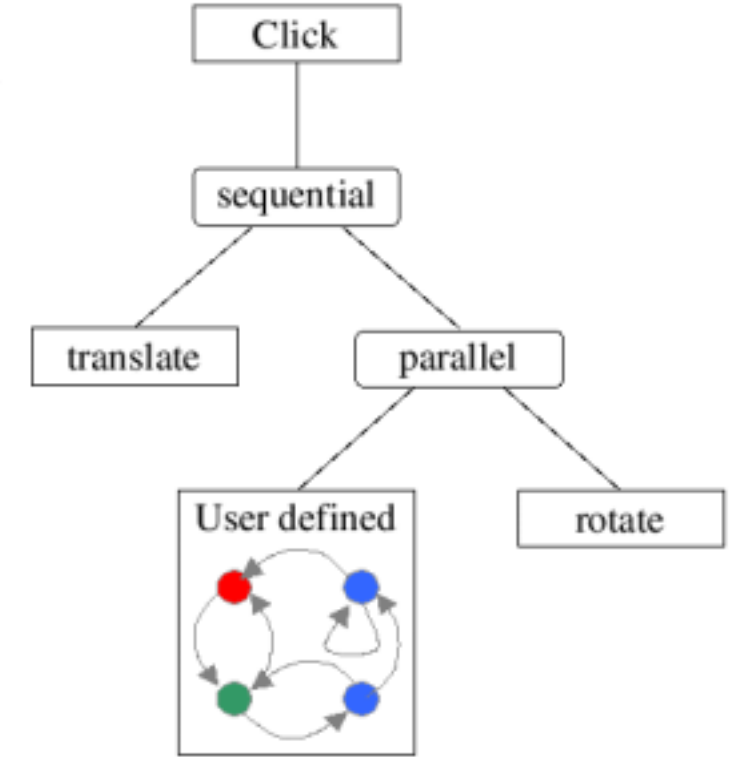


**Fig. 5.12** Ordonnancement des comportements dans un graphe.

```
  buttonPressed AFFECTS monOrdonnanceur.start
}

DEF monOrdonnanceur SequentialAnim {
  children [
    ParallelAnim {
      children [
        RotationInterpolator {
          key [...]
          keyValue [...]
          value_changed AFFECTS monObjet.set_rotation
        }
PostionInterpolator {
          key [...]
          keyValue [...]
          value_changed AFFECTS monObjet.set_translation
        }
      ]
    }
```

```
    RotationInterpolator {
      key [...]
      keyValue [...]
      value_changed AFFECTS monObjet.set_rotation
    }
  ]
  key [...]
  keyValue [...]
  value_changed AFFECTS monObjet.set_rotation
}
```

Les nœuds de l'ordonnanceur remplacent les nœuds *TimeSensor*. Les animations sont effectuées une seule fois et chacune d'elles est exécutée jusqu'au bout (jusqu'à la durée indiquée par le concepteur de la scène dans les champs *key* des interpolateurs). Dans l'exemple précédent, si la première rotation dure 2 secondes et la translation 5 secondes, alors la seconde rotation s'exécutera après 5 secondes. Si celle-ci dure 10 secondes, l'ensemble de l'animation durera donc 15 secondes.

### *Les comportements 2D*

Une fonctionnalité importante des *widgets* est la possibilité de les réunir dans un *widget* conteneur. Avec les classes AWT (*Abstract Window Toolkit*) de Java, on ne les positionne pas de manière absolue mais par une position relative grâce à une politique de placement associée au conteneur. Cette abstraction des transformations est intéressante pour les non-programmeurs et pour contraindre un objet à rester dans le volume défini par son conteneur.

Dans une application programmée avec un langage objet, l'implémentation des comportements et des politiques de placement est rendue plus facile par l'utilisation de classes. La figure 5.13 schématise les classes et méthodes nécessaires pour implémenter un bouton composé d'un cylindre et d'une sphère posée sur celui-ci. Pour les comportements, les classes correspondent à des classes du type de celles de Java3D. Pour les politiques de placement, elles ressemblent aux classes AWT de Java mais avec des positionnements 3D. Au lieu de "North", "South", "East", "West" et "Center" pour la classe *BorderLayout*, des couples sont employés en associant les précédents pour le positionnement dans un plan horizontal avec "Up", "Middle", "Down" pour le positionnement en profondeur. L'étude de ces classes est cependant intéressante pour permettre d'enrichir l'ensemble des *widgets* 2D actuelles avec de nouvelles en 3D. Cette perspective est particulièrement séduisante pour continuer notre étude sur l'intégration 2D/3D.

Les autres politiques de placement 2D incluses dans l'AWT [Flanagan96] (dans la table 5.1) ont été traduites en leur équivalent 3D, apportant pour certaines quelques propriétés déjà utilisées dans la 3D.

Ci-dessous, est donnée la spécification en VRML du nouveau nœud *BorderLayout3D*. Les autres politiques de placement sont spécifiées de la même manière. Ils sont gérés en prenant en compte les boîtes englobantes de leurs nœuds enfants. L'échelle des enfants est réduite

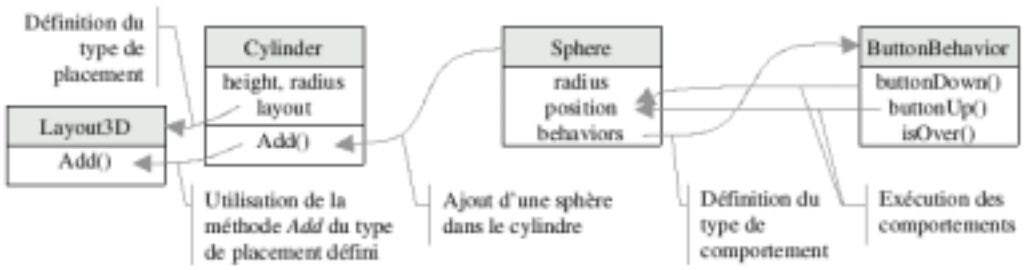


**Fig. 5.13** Politique de placement et comportements programmés.

| **Layout** | *Description* |
|---|---|
| **CardLayout** | Assure qu'un seul objet graphique est visible à la fois : c'est l'équivalent d'un nœud *Switch* en VRML et elle permet de spécifier des géométries différentes pour assurer un changement de niveau de détails. |
| **FlowLayout** | Aligne les composants comme les mots sur une page (quand la ligne est pleine elle recommence sur une nouvelle) : les trois dimensions seront utilisées dans l'ordre voulu pour obtenir le même effet. |
| **GridBagConstraints** | Associée à *GridBagLayout*, offre un plus grand contrôle du placement d'un composant (on peut donner sa position dans une cellule, ses marges). |
| **GridBagLayout** | Divise le conteneur en lignes et colonnes comme *GridLayout* mais permet d'avoir des largeurs et hauteurs différentes pour chaque composant. |
| **GridLayout** | Dispose les objets dans une matrice : en 3D l'ajout d'une troisième dimension à cette matrice permettrait d'assembler n'importe quel objet. |

**Tab. 5.1** Les politiques de placement (*layouts*) de l'AWT de Java.

jusqu'à ce qu'ils s'adaptent et entrent dans le conteneur qui a une taille fixe, spécifiée par le champ *bboxSize*.

```
BorderLayout3D {
  field        SFVec3f    bboxCenter
  field        SFVec3f    bboxSize
  exposedField SFNode     upNorth       NULL
```

```
  exposedField SFNode     upCenter      NULL
  exposedField SFNode     upSouth       NULL
  exposedField SFNode     upEast        NULL
  exposedField SFNode     upWest        NULL
  exposedField SFNode     middleNorth   NULL
  [...]
  exposedField SFNode     middleWest    NULL
  exposedField SFNode     downNorth     NULL
  [...]
  exposedField SFNode     downWest      NULL
}
```

Ces manières de positionner les objets peuvent parfaitement s'ajouter à la liste des nœuds de regroupement. Ils se substituent aux nœuds *Group* ou *Transform*. L'ordre dans lequel sont données les différents objets dans un tel nœud influe directement sur leur placement. Dans un nœud comme *BorderLayout3D*, l'équivalent 3D de *BorderLayout*, les éléments sont rassemblés dans autant de champs qu'il n'y a de points cardinaux 3D.

Le nœud *GridLayout3D* dispose, comme on peut le voir dans sa spécification donnée ci-dessous, d'un unique champ *children* dans lequel seront placés les nœuds enfants. La boîte englobante du conteneur sera divisée pour obtenir le nombre de cases nécessaires (*XSize*YSize*ZSize* en l'occurrence). Les enfants adapteront leur échelle et occuperont dans l'ordre toutes les colonnes, puis toutes les rangées puis toutes les profondeurs. Dans les deux cas donnés en exemple, une transformation n'est alors plus nécessaire pour positionner ces objets.

```
GridLayout3D {
  field        SFVec3f    bboxCenter
  field        SFVec3f    bboxSize
  field        SFInt32    XSize
  field        SFInt32    YSize
  field        SFInt32    ZSize
  exposedField MFNode     children      []
}
```

### *Les comportements 3D*

Le succès à large échelle de VRML repose sur une description des animations et des comportements aussi compréhensible que celle des objets. Cette simplification implique de réduire la place réservée aux scripts gérant les interactions. Nous introduisons pour cela des nœuds de comportements 3D applicables à tout objet. Nous optons également pour le remplacement des routages par des mécanismes simples de contrôle des liens entre nœuds.

Avec notre solution, nous ne remettons pas en cause le modèle de description que nous trouvons puissant et proche des API 3D récentes (telles que Java3D, Direct3D, Fahrenheit). Cependant, pour pallier l'inexistence d'un modèle haut niveau pour décrire les comportements nous l'augmentons sans pour autant dénaturer sa structure.

Pour les scènes qui nous servent d'exemples, nous avons souhaité les extensions suivantes.
- Un nœud *Clickable* gérant les clics de souris et fournissant les trois apparences classiques : appuyée, relâchée et *roll-over*. Il permettrait de gérer l'affichage des étiquettes informatives et l'interaction menant à l'ouverture d'un livre. La définition du nœud en VRML est la suivante :

```
Clickable {
  eventIn      MFNode     addChildren
  eventIn      MFNode     removeChildren
  exposedField MFNode     children        []
  eventOut     SFTime     buttonPressed
  eventOut     SFTime     buttonReleased
  eventOut     SFBool     isOver
}
```

- Un nœud *Movable* signifiant que les interactions de l'utilisateur affectent ses nœuds descendants. Ce nœud permettrait de positionner et d'orienter les objets grâce à des interactions et non plus par navigation comme précédemment. En effet, pour déplacer une seule fenêtre dans le poste de lecture en VRML+Java, il faut punaiser toutes les fenêtres sauf celles que l'on souhaite déplacer. La définition du nœud est la suivante :

```
Movable {
  eventIn      MFNode     addChildren
  eventIn      MFNode     removeChildren
  exposedField MFNode     children        []
  exposedField SFBool     canTranslate TRUE
  exposedField SFBool     canRotate TRUE
  eventOut     SFvec3f    translation
  eventOut     SFRotation rotation
}
```

- Un mot clé *AFFECTS* pour éliminer la partie routage du fichier VRML. Lorsque le nœud de comportement ne modifie que ses nœuds descendants, ce mot clé n'est pas nécessaire. Il l'est pour intégrer directement dans le graphe de scène les routages d'événements vers un nœud non descendant (comme dans le code suivant).

Pour avoir une idée plus précise du code VRML obtenu, le pseudo-code ci-dessous décrit une fenêtre 3D avec les nœuds précédents. Cette nouvelle syntaxe remplace 450 lignes de Java servant à gérer les fenêtres et le fichier VRML ne dépasse pas 125 lignes.

```
DEF Window3D Transform {
  children [ # Les noeuds constituant la fenêtre
    Shape { [...] }
    [...]
    Shape { [...] }
    DEF TitleBar Movable {
      # Les interactions autorisées
      canTranslate TRUE
      canRotate TRUE
      # Les événements émis par ce noeud
      translation AFFECTS Window3D.translation
```

```
      rotation AFFECTS Window3D.rotation
      children [...] # Les noeuds constituant la barre
    }
  ]
}
```

L'utilisation du mot-clé *AFFECTS* permet de clarifier les liaisons entre objets et comportements dans la scène. Il n'est, en effet, pas nécessaire d'aller voir en fin de fichier et chercher le routage d'événement que l'on souhaite modifier.

### Description des comportements

Nous remarquons que la majorité des applications 3D implémentent presque toujours le même ensemble d'interactions (clique de souris pour la sélection, détecteur de proximité ou de collision, ...). Cependant, les comportements prédéfinis, et par conséquent génériques pour pouvoir être utilisés dans le plus grand nombre d'applications possibles, est bien trop limitante.

La solution que nous préconisons se compose de nœuds génériques de comportements et d'un langage de description pour créer des comportements supplémentaires. Dans la figure 5.7 page 201, nous avons schématisé les nœuds *Transform* en leur fournissant une partie qui rassemble les primitives graphiques pour construire un objet (par le champ *children*) et une partie qui permet d'attacher des comportements (par l'intermédiaire du champ *behavior*). Le nœud *Clickable* rend sensible aux clics de souris, les nœuds présents dans son champ *children*. Plus précisément, le déplacement de la souris avec son bouton appuyé effectue une rotation de l'objet. L'action *isOver* déclenche, quant à elle, l'exécution d'un script en Java qui opère un changement de l'apparence de l'objet par exemple. Nous voyons donc que les nœuds de comportements, bien que susceptibles d'effectuer les interactions les plus souvent utilisées, ne permettent pas de supprimer les scripts. Un autre problème se pose : Quel langage choisir pour spécifier les comportements capables de modifier ou créer une apparence ?

La syntaxe d'un langage comme Alice est trop proche de celle des langages objets. Il nécessite l'installation d'un interpréteur Python et ce n'est pas un langage que les modeleurs spécialisés pourront générer facilement. Les automates présentés par Zeleznik, Herndon, Conner et associés [Conner92, Zeleznik91] permettent de créer une multitude de comportements. Ceux donnés en illustration dans leurs articles servent plutôt à la construction de *widgets* qui traitent principalement de la navigation. Cependant, leur système convient pour spécifier des interactions évoluées comme celles pour manipuler un arbre conique décrit dans [Robertson93a]. La spécification de ces automates pour les comportements inter-objets peut être donnée à l'aide d'un langage, comme nous l'illustrons dans la figure 5.14.

Dans cet exemple, les objets *MAT* et *OBJ* doivent être définis dans le graphe VRML par des mots-clés *DEF*. La fonction *move*, quant à elle, est une fonction prédéfinie, gérée par le navigateur, permettant de calculer la translation de l'objet *OBJ* en fonction de la translation *trans* du pointeur. Le problème avec ces automates et leur langage associé est que l'on

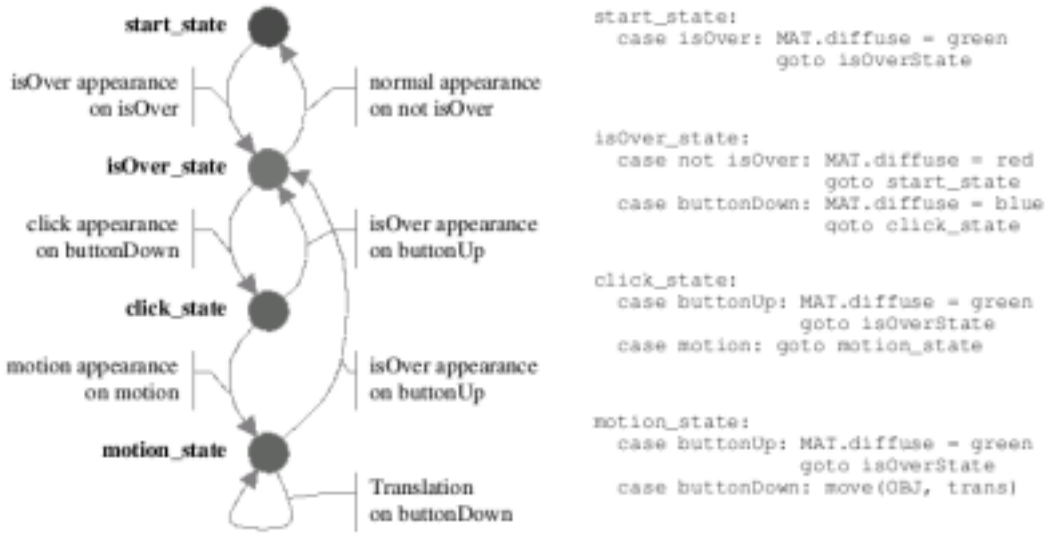


**Fig. 5.14** Automate de comportements.

ne peut pas faire abstraction du périphérique de pointage utilisé. Dans l'automate précédent, nous avons volontairement utilisé des termes génériques. Cependant, le sens donné à *isOver* convient plus facilement à un pointeur 2D. Néanmoins, nous laissons l'interprétation de ces mots-clés et la gestion des périphériques à la charge du navigateur. Les boites englobantes des objets pourront lui servir à gérer de tels événements avec un pointeur 3D : si le pointeur se trouve dans la boite englobante d'un objet alors le navigateur considère que "le pointeur *isOver* l'objet".

A l'heure actuelle, seules les interactions avec la souris 2D sont prises en charge. Nous reviendrons, dans la conclusion page 221, sur les solutions que nous envisageons d'introduire dans notre navigateur VRML pour prendre en charge tous les périphériques de pointage.

Pour inclure cette description des interactions dans un graphe de scène VRML, on a introduit un nœud *UserBehavior* dont un champ de type *MFString* (chaînes de caractères multiples) donne les différents états de l'automate. Cette solution, contrairement à un langage VRML contraint [Balaguer99, Diehl00], n'empêche pas la dé-corrélation entre les deux moteurs puisque les automates sont uniquement attachés à ce type de nœud de comportement. En aucun cas, leur exécution n'est dépendante des nœuds de l'arbre mais uniquement des événements reçus.

### 5.3.2 Son

Nous avons expliqué pour quelles raisons nous avions choisi VRML pour décrire les scènes sonores. Tout langage de description 3D aurait pu convenir. Mais, VRML est le seul langage dont la spécification est claire. C'est aussi, malgré ses lacunes, le langage le plus utilisé, le format avec lequel on trouve le plus de scènes décrites. Enfin, et c'est ce qui a véritablement motivé notre choix, nous avons étudié VRML97 depuis ses débuts, en 1997, alors qu'il

n'était pas encore une norme ([TOPOL97]). Cela nous assure une bonne connaissance de sa syntaxe et de ses possibilités.

Les extensions que nous avons proposées ont été implémentées en utilisant l'un de nos deux navigateurs VRML développés au CNAM, celui écrit en C. La syntaxe des nouvelles expressions incluses a été ajoutée à la syntaxe VRML. Les nouveaux nœuds ont été pris en charge comme tout autre nœud, par un code propre. S'agissant de nœuds sonores, les fonctions d'OpenAL ont été utilisées à la place de fonctions OpenGL. Nous détaillons l'implémentation du prototype à partir de la page 217.

Les extensions décrites dans les prochaines pages sont issues de l'étude des possibilités actuelles de VRML pour la gestion des sources sonores spatialisées. Elles visent à compléter la spécification VRML qui, come nous l'avons vu page 197, ne définit que trois nœuds pour rendre une scène sonore : *Sound*, *AudioClip* et *MovieTexture* (voir la figure 4.33 page 183).

## EXTENSIONS

Les attributs sonores que nous avons ajoutés à VRML l'ont été en considérant la ressemblance entre les sons et les graphiques 3D. De cette manière, nous avons pu utiliser une même sémantique pour les deux aspects. Plus exactement, nos nouveaux nœuds sonores utilisent la même logique que les nœuds graphiques déjà présents. Les aspects que nous avons étudiés de près pour nos extensions sont :
- le type de sources sonores nécessaires,
- les nœuds de groupement pour définir les effets de salle,
- les attributs sonores des géométries.

### *Sources Sonores*

Une étude comparative entre les synthèses sonores et graphiques 3D a été faites dans [NATKIN98]. Une des différences majeures relevée, est la plus grande directivité des rayons lumineux. Cependant, il peut être intéressant, dans un monde virtuel, de créer des ondes sonores très directionnelles, pour obtenir des effets spéciaux. Alors que l'on a besoin de lumières de type spot, point et directionnelle pour obtenir des effets visuels différents, pour les sons, un seul modèle nous suffit pour créer de nombreux effets.

Un son décrit à l'aide de cônes peut être utilisé pour simuler différentes "formes", de la plus réaliste (omnidirectionnelle) à la plus irréaliste (unidirectionnelle). Deux cônes seront nécessaires pour définir les régions dans lesquelles le son est perçu avec une intensité maximale, une intensité réduite et une intensité nulle. La manière de calculer l'atténuation du son est la même qu'avec les ellipses.

Le nœud VRML correspondant, que nous avons appelé *SoundCone*, afin de conserver le nœud *Sound* pour une compatibilité avec les scènes VRML standards, est défini de la manière suivante :

```
SoundCone {
  exposedField SFVec3f location       0 0 0
```

```
  exposedField SFVec3f direction      0 0 1
  exposedField SFFloat innerAngle     6.2832
  exposedField SFFloat outerAngle     6.2832
  exposedField SFFloat outerAngleGain 0.0
  exposedField SFFloat presence       0.0
  exposedField SFFloat warmth         0.0
  exposedField SFFloat brilliance     0.0
  exposedField SFNode  source         NULL
  field        SFBool  spatialized    TRUE
}
```

Les valeurs par défaut pour les deux cônes sont de 360 degrés, c'est-à-dire que par défaut les sources sonores sont omnidirectionnelles. On pourra toutefois modifier ces paramètres pour créer des sources sonores directionnelles. Les attributs *presence*, *warmth* et *brilliance* sont des variables pour modifier le timbre du son. Elles sont utilisées dans le spatialisateur de l'IRCAM et dans EAX. Les autres champs sont issus de la spécification du nœud *Sound* et permettent d'attacher le son à jouer (*source*), et de donner sa position (*location*) et son orientation (*direction*).

### *Effets de salle*

Deux solutions peuvent être considérées pour intégrer les effets de salle au modèle VRML. Les variables nécessaires pour simuler les effets de réflexion peuvent être définies dans un nœud général (*binding node*) ou dans un nœud de groupement. Dans le premier cas, nous ne pourrions attribuer qu'un seul effet de salle pour toute la scène. Cela ne serait d'aucune utilité pour gérer les scènes décrites pour le "tapis persan" [Natkin00, Natkin01]. On doit pouvoir définir plusieurs environnements sonores indépendants dans ce projet. Même si la scène globale était définie en utilisant plusieurs fichiers VRML, un seul des *binding nodes* pourrait être activé à la fois.

Un nœud de groupement est beaucoup plus puissant puisque plusieurs effets de salle peuvent être définis dans une même scène. Les effets de salle sont actifs lorque l'utilisateur se trouve à l'intérieur de la boîte englobant toutes les géométrie descendant du nouveau nœud de groupement. Cette boîte englobante, définissant l'espace dans lequel l'effet de salle est actif, peut être définie par le concepteur de la scène à travers les attributs *bboxCenter* et *bboxSize*, pour donner respectivement le centre et la taille de la boîte.

VRML compte plusieurs nœuds de groupement, chacun d'eux exécutant une opération spécifique sur leurs nœuds descendants :
- *Anchor* spécifie les nœuds qui agissent comme des liens,
- *Billboard* rassemble les géométries qui doivent être orientées automatiquement vers l'utilisateur,
- *Collision* décrit les paramètres de collision pour ses nœuds enfants,
- *Inline* lit dans une URL externe la description de ses enfants,
- *LOD* définit différents niveaux de détails pour les objets et fournit des informations pour que le navigateur puisse choisir le bon rendu à employer en fonction de la distance,
- *Switch* choisit 0 ou 1 nœud à afficher parmi de ses enfants,

– *Transform* définit un repère local pour ses nœuds enfants.

Pour conserver cette approche, c'est-à-dire à un nœud de groupement équivaut une opération spécifique, nous avons défini un nouveau nœud pour spécifier les effets de salle. Ce nœud, que nous avons appelé *RoomEffect*, définit les variables globales à utiliser pour la synthèse psycho-acoustique lorsque l'utilisateur se trouve dans sa boîte englobante. Ces variables sont, encore une fois, issues du spatialisateur et de l'API sonore EAX. La définition du nœud VRML correspondant est la suivante :

```
RoomEffect {
  eventIn      MFNode     addChildren
  eventIn      MFNode     removeChildren
  exposedField MFNode     children        []
  exposedField SFBool     doplerEffect    TRUE
  exposedField SFBool     airAbsorbtion   TRUE
  exposedField SFFloat    roomPresence    0.0
  exposedField SFFloat    runningReverb   0.0
  exposedField SFFloat    envelope        0.0
  exposedField SFFloat    lateReverb      0.0
  exposedField SFFloat    heaviness       0.0
  exposedField SFFloat    liveness        0.0
  field        SFVec3f    bboxCenter      0 0 0
  field        SFVec3f    bboxSize        -1 -1 -1
}
```

La boîte englobante est calculée automatiquement en fonction des géométries enfants, dès lors qu'elle n'est pas explicitement spécifiée. Ce calcul automatique peut toutefois créer des boîtes englobantes qui se pénètrent si leurs nœuds enfants ne sont pas correctement spatialisés. Dans ce cas, le premier nœud *RoomEffect* dans lequel l'utilisateur se trouve, sera choisi.

### *Filtrages sonores*

Les attributs sonores, ou matériaux sonores, des objets sont pour l'instant très simples. Nous n'avons intégré que l'atténuation du volume des sons. D'autres filtres pourraient être décrits, comme des filtres passe-haut ou passe-bas.

Les attributs visuels sont fournis pour les deux côtés. Avec le nœud *IndexedFaceSet* (donnant une série de polygones), il est possible de spécifier si les deux faces doivent être dessinées. Si c'est le cas, la même apparence sera utilisée pour les deux. Avec les autres primitives géométriques (*Box*, *Cone*, *Cylinder* et *Sphere*), les deux faces sont toujours dessinées de la même manière.

Pour les attributs sonores, nous ne pouvons pas utiliser le même principe puisque nous voulons avoir des effets différents en fonction du sens avec lequel le son traverse les géométries. C'est pourquoi, nous avons défini des paires d'attributs, pour chaque sens (pour la direction extérieur-intérieur et la direction opposée). Avec les géométries décrites à l'aide

du nœud *IndexedFaceSet*, la face externe sera celle de laquelle sortent les normales. Si uniquement un côté des faces doit être dessiné, les sons ne seront filtrés que dans un sens. Tous les calculs d'intersection nécessaires pour déterminer les intersections entre un son et des géométries sont faits à l'aide d'algorithmes empruntés au lancé de rayons.

Le code suivant fournit un exemple du nouveau nœud *SoundFiltering*. Il peut être attaché à un nœud *Shape* décrivant une géométrie et son apparence. On adjoint donc aux propriétés visuelles des géométries, des propriétés sonores. Les attributs *frontIntensity* et *backIntensity* sont utilisés pour spécifier le pourcentage de l'intensité de chaque son qui est filtrée. Cette manière de décrire le filtrage est la même que celle utilisée pour le champ *transparency* dans le nœud *Material*.

```
[...]

SoundCone { # la première source sonore
  location 0 -10 0
  direction 0 0 -1
  source AudioClip {
    url "bipbip.wav"
  }
}

[...] # les autres sources sonores

RoomEffect { # première salle
  bboxCenter 0 -8 0
  bboxSize 3 3 2
  children [
    Shape { # premier mur de la salle
      geometry IndexedFaceSet {
        [...] # les polygones définissant le mur
      }
      appearance Appearance {
        [...] # les propriétés visuelles du mur
      }
      soundFiltering SoundFiltering {
        frontIntensity 0.5
backIntensity 0.8
      }
    }
    Shape { # deuxième mur
      geometry [...]
      appearance [...]
      soundFiltering SoundFiltering {
        frontIntensity 0.3
backIntensity 0.7
      }
```

```
    }
    [...] # les autres murs
  ]
}

RoomEffect { # la deuxième salle
  bboxCenter 0 -4 0
  bboxSize 3 5 2
  children [
    [...]
  ]
}
[...]
```

La figure 5.15 correspond à la représentation de la scène décrite par le code précédent. Les zones blanches identifient les positions où l'utilisateur ne percevra aucun son. Les dégradés de gris correspondent aux zones sonores. Plus la zone est foncée, plus les sons seront filtrés par les géométries.

## Implémentation

Pour l'implémentation de ces principes, nous avons utilisé notre navigateur VRML écrit en C [Topol01]. Pour que notre navigateur puisse lire les fichiers VRML augmentés de ces nouveaux nœuds, nous avons défini les ajouts sous une nouvelle version de VRML (avec un nouvel entête de fichier : `#VRML V3.0 utf8`). La modification des grammaires Yacc et Lex pour y ajouter les nouveaux nœuds et attributs, n'a pas posé de problèmes particuliers. Nous avons fait attention, lors de la spécification des nouveaux nœuds, à utiliser une syntaxe très proche des autres nœuds.

Pour l'interprération des nœuds sonores, nous avons également conservé la manière de traiter les nœuds usuels. Néanmoins, au lieu d'utiliser des commandes d'OpenGL ou de Direct3D, nous avons bien évidemment fait appel à des fonctions d'une API sonore, en l'occurrence, OpenAL. Les commandes d'OpenAL son très proches de celles d'OpenGL, ce qui nous a permis un apprentissage rapide. Par exemple, pour positionner une source sonore, il suffit d'utiliser la fonction `alSourcef` qui attend les mêmes paramètres que la fonction `glTranslatef`, c'est-à-dire, les trois composantes x, y et z. De la même manière, l'auditeur est positionné à l'aide de la fonction `alListenerf`. De plus, les *buffers* de sons avec OpenAL se gèrent de la même manière que les *buffers* de textures avec OpenGL.

Le seul léger problème vient du fait que les commandes d'OpenAL ne sont pas assujetties aux transformations d'OpenGL. Puisque les sons dans VRML doivent subir les transformations, nous avons donc du calculer les positions des sources sonores, dans le repère global de la scène, en prenant en compte la matrice de transformation d'OpenGL. De même, pour animer les sources sonores par des nœuds d'interpolation, il a fallu calculer la matrice composée par toutes les transformations, y compris celles de l'animation, et l'appliquer à la source sonore en mouvement. Pour les déplacements de l'utilisateur, les positions sont plus

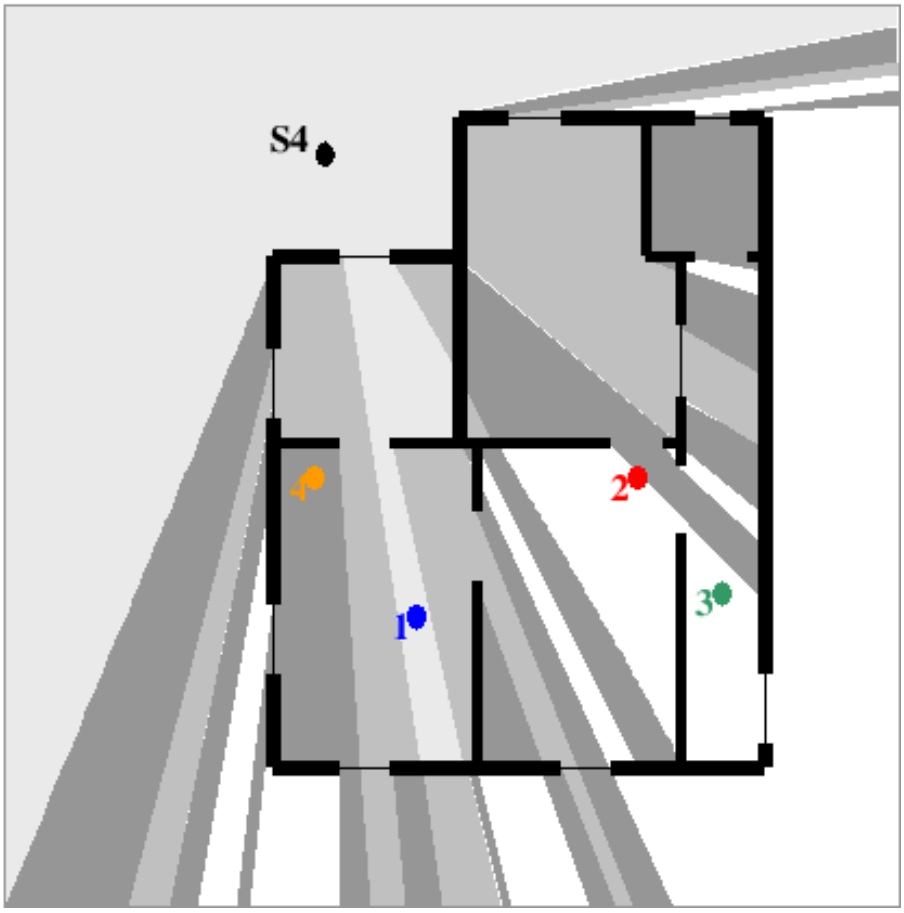


**Fig. 5.15** Exemple d'absorption sonore des géométries.

faciles à prendre en compte car, en VRML, ils sont donnés dans le repère global (comme avec le nœud *Viewpoint*).

L'implémentation des effets de salle a été faite à l'aide de la librairie EAX. L'occlusion, quant à elle, a nécessité l'utilisation d'algorithmes proches de ceux employés pour le lancé de rayons. Il a fallu, pour chacune des sources sonores, calculer les intersections entre la droite la liant à l'utilisateur, et les géométries[1]. Tous les polygones coupés par cette droite sont susceptibles alors de filtrer le son. Pour déterminer quel filtrage utilisé, *frontIntensity* ou *backIntensity*, il suffit de déterminer l'angle entre le "rayon sonore" et la normale au polygone.

Les calculs permettant de définir les sons perçus, doivent être faits à chaque fois qu'un attribut entrant dans les opérations, est modifié. C'est le cas, notamment, des mouvements de caméra (équivalents aux déplacements de l'utilisateur), des mouvements des sources sonores et des animations des objets. La chaîne de production d'un son binaural est décrit dans la figure 5.16. Les effets de salle ne sont pas complètement implémentés dans le prototype, car ils ne sont pas présents dans OpenAL et n'étaient pas très développés dans

[1] L'occlusion est dorénavant gérée dans la version 2 d'EAX qui n'était pas alors disponible.

EAX 1.0. C'est pourquoi, nous avions alors pensé à utiliser notre deuxième navigateur, écrit en Java, pour bénéficier des apports du Spatialisateur de l'IRCAM. Cependant, il semblerait qu'une solution utilisant EAX 2.0 soit aujourd'hui préférable, sachant que les effets de salle et les occlusions y sont bien supportés.

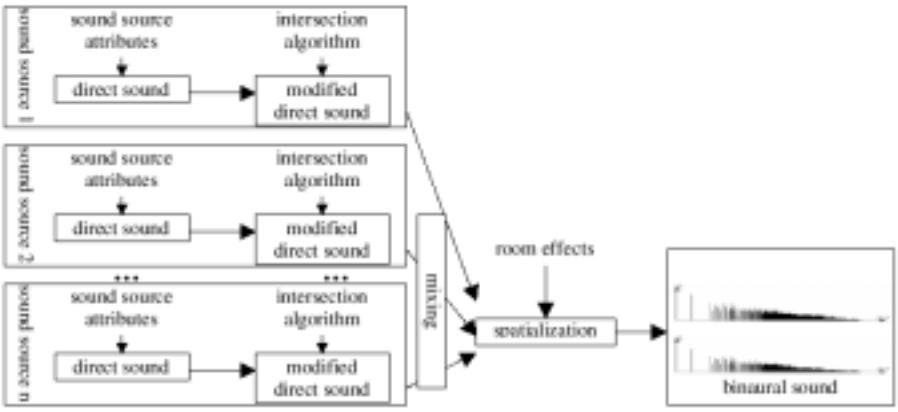


**Fig. 5.16** La chaîne de production binaurale des sons.

# Chapitre 6

# *Conclusion et travaux futurs*

## Constats techniques

Notre étude des interfaces 3D regroupe différentes facettes qui bien que très différentes sont toutefois indissociables. En premier lieu, un apprentissage des étapes et techniques menant d'une description de scène à son affichage a été nécessaire. Pour l'aspect description de la scène (si celle-ci n'est pas créée entièrement par un programme mais plutôt lue dans un fichier support), cela suppose la maîtrise du langage utilisé (lorsque l'on n'utilise pas un outil spécialisé). Dans notre cas, nous avons choisi VRML97 et nous avons poussé son étude jusqu'à l'écriture de deux interpréteurs, qui traduisent les expressions reconnues en appels de fonctions d'une API 3D. Pour cela, il a donc été également nécessaire d'étudier les API 3D afin d'effectuer l'affichage des scènes lues. Ces travaux ont été présentés dans les deux mémoires d'ingénieur CNAM [Abisror02, Topol01].

Les aspects précédents gravitent exclusivement autour de la spécification et de la conception graphiques des interfaces 3D. On entend par cela, la construction d'objets à partir de primitives graphiques 3D et leur rendu lors de l'affichage. Or, pour toute application exploitant une interface 3D, les interactions sont également d'une extrême importance. Alors qu'une interface 3D fournit à l'utilisateur une nouvelle manière d'organiser et de voir les informations, les interactions lui offrent les outils pour la manipuler, y naviguer et par conséquent de mieux appréhender les informations qu'elle renferme. Trouver des méthodes de manipulation et de navigation fluides et intuitives est aussi important que de trouver les métaphores visuelles adéquates pour révéler les informations.

Il nous apparaît que les méthodes d'interaction et de navigation et celles de représentation des objets sont indissociables. Elles aident à part égale à appréhender et à analyser les informations renfermées dans une interface. Une interface 3D nous offre cette propriété

quasi exclusive de pouvoir utiliser nos expériences réelles sur des objets virtuels sous réserve que des interactions particulièrement naturelles soient également implémentées.

De cette synthèse sur les techniques et technologies 3D, deux constats peuvent d'ores et déjà être tirés. Premièrement, les interfaces 3D ont bénéficié, depuis une dizaine d'années, de recherches poussées au niveau des algorithmes de rendu et de leur utilisation. Deuxièmement, en comparaison des techniques de rendu, les interactions 3D ne sont étudiées que depuis récemment et de manière très diffuse. En d'autres mots, de nombreuses équipes de recherche ont démontré l'utilité de la visualisation 3D sans s'intéresser (ou peu) à son utilisation. Il en résulte un terrible décalage entre ce qu'il est possible de représenter grâce à une interface 3D et ce que l'utilisateur novice peut appréhender par une manipulation hasardeuse à travers des dispositifs inadéquats. De plus, aucune facilité n'existe pour aider le programmeur à implémenter les interactions : aucune API n'est fournie pour faire correspondre les mouvements des dispositifs de pointage avec ceux des objets 3D.

Cela ne remet néanmoins pas en cause l'efficacité apportée par les techniques 3D pour la représentation de nombreuses données. Le nombre de projets de visualisation en 3D, que nous avons rassemblés sous le nom de "paysages informationnels" dans le chapitre 2, illustre l'intérêt porté aux nouvelles possibilités offertes par la 3D. Nous soulignons ici les lacunes en matière d'interactions 3D comparées aux possibilités de rendu des scènes. Notre contribution se place précisément à ce niveau dans le cadre de l'étude du langage VRML.

## Applications

Dans un deuxième temps, nous avons mis à profit nos études des API 3D et du langage VRML pour créer divers projets d'interfaces 3D. Les premières s'attachent à la réalisation de deux des trois activités qu'un lecteur accomplit dans une bibliothèque : la recherche d'ouvrages et la lecture. Le deuxième projet est le prototypage d'un environnement 3D partagé entre applications 2D et 3D.

Pour ces différents projets, une interface s'appuyant sur des techniques 3D temps réel, permet de faire massivement usage des textures pour mettre en place des heuristiques visuelles. Celles-ci pourraient être implémentées avec une interface 2D classique. Mais la troisième dimension et le zoom, naturellement mis à disposition par le changement de point de vue, ouvrent de nouvelles possibilités d'organisation et d'interaction avec les documents. Ces caractéristiques réunies font de la 3D temps réel une technologie mieux adaptée que la 2D pour construire des interfaces visuelles rassemblant de nombreuses informations.

Une interface zoomable aurait également tout à fait pu convenir pour bénéficier d'une troisième dimension et d'un zoom interactif. Mais, dans la perspective d'utiliser l'espace pour organiser les informations, les rotations offertes par la 3D, et non présentes dans les interfaces 2.5D, sont très utiles voire nécessaires. Alors qu'avec une interface 2.5D, les objets ne peuvent qu'être répartis en profondeur sur différents plans parallèles, avec une interface 3D, ils peuvent également être orientés vers l'utilisateur. Or, il nous semble beaucoup plus naturel d'orienter son point de vue vers les données plutôt que de le translater. Transposé dans le monde réel, cela revient à orienter son corps pour regarder une autre portion de l'espace plutôt que de devoir se déplacer latéralement (figure 6.1). Cette intuition, née de ce

rapprochement entre univers réel et virtuel et sur laquelle nous nous sommes appuyés, nous paraissait tellement évidente que nous ne l'avons p ! as validée. Une étude pour comparer l'efficacité des deux méthodes de modification du point de vue pour pointer vers les données pourrait être réalisée.

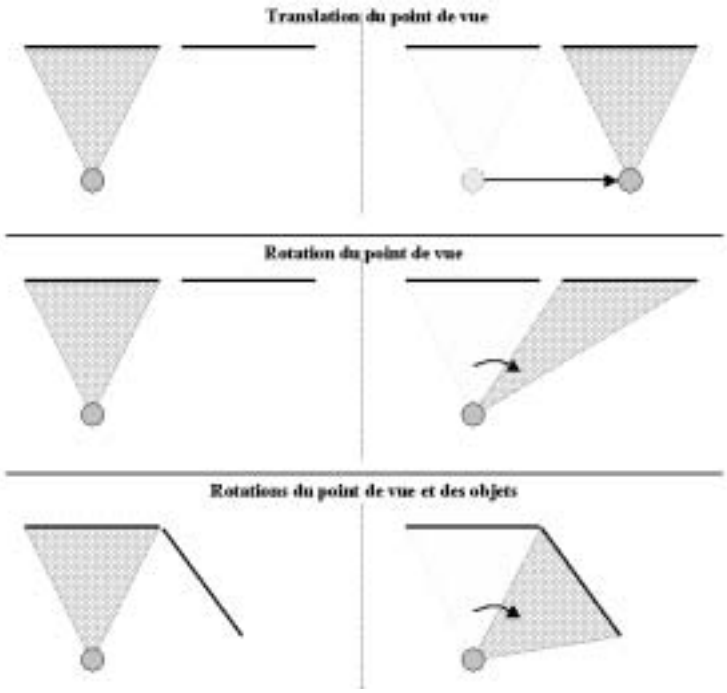


**Fig. 6.1** L'apport des rotations dans une interface 3D.

## Bibliothèques numériques

Même si nous manquons encore de recul dans l'observation des usages, il semble que les bibliothèques numérisées actuellement en service n'offrent pas un confort d'utilisation suffisant pour dépasser le rôle bien restreint de diffusion de fac-similés pour l'impression à distance. Pour les concepteurs de ces services, la question de l'ergonomie du poste de lecture a pu paraître mineure devant d'autres problèmes comme le traitement des images numérisées, leur stockage et leur transmission jusqu'à l'usager (en se limitant aux questions strictement informatiques). Ces questions sont progressivement passées au second plan, mais il a fallu également tenir compte des limitations du WWW qui s'est imposé comme le support de ces interfaces. Les évolutions constantes de ce standard, ainsi que la banalisation des cartes graphiques permettant la production d'images synthétiques en temps-réel ouvrent de nouvelles opportunités en matière d'interaction humain-machine que nous avons commencé ! à étudier.

Dans les bibliothèques numériques actuelles, le mode "parcours" est toujours associé à un moteur de recherche - une fonction jugée indispensable dès lors que le corpus atteint une certaine taille. Aussi sophistiqué soit-il, ce type de fonctionnement n'a d'intérêt que

pour l'usager muni d'un projet préconçu : il ne permet en aucun cas de flâner dans une bibliothèque numérique. Il est assez simple de compléter les interfaces strictement textuelles, habituellement rencontrées, par un fonctionnement beaucoup plus graphique, en numérisant les dos d'ouvrages à leur dimensions respectives.

Les interfaces de bibliothèques numérisées souffrent très rapidement du problème d'encombrement bien connu des interfaces WIMP, dès lors que l'usager se met en phase de lecture active. Dans leur article très remarqué décrivant le *Web Forager* [CARD96], Card et Robertson (Xerox) ont préconisé l'utilisation d'un espace synthétique 3D pour exploiter les effets de zoom, de mise en perspective et de déplacement du point de vue. Ces techniques d'interaction 3D peuvent aussi être appliquées lors de la manipulation du document lui-même. L'acte de feuilletage est alors simulé par une interface 3D associée à un écran tactile ou un pointeur conventionnel. L'utilisation de la navigation 3D est parfois remise en cause, au profit par exemple de systèmes de zoom hiérarchique et le débat n'est pas clos [PLÉNACOSTE01].

L'architecture interne du *Web Forager* n'ayant pas fait l'objet de description, nous avons repris ses idées avec une interface s'appuyant sur VRML. Le choix d'utiliser ce langage s'est imposé naturellement car nous voulions décrire nos interfaces rapidement en nous appuyant sur une norme capable d'être utilisée sur Internet. Au début du projet, VRML se présentait comme le seul langage réunissant ces critères.

Deux interfaces ont été implémentées avec le langage VRML, l'une pour la recherche [CUBAUD98] et l'autre pour la consultation [CUBAUD00, CUBAUD01]. Pour la première, VRML n'a servi qu'à spatialiser les résultats des recherches soumises par l'intermédiaire d'un formulaire banal. Nous avons donc utilisé uniquement les capacités descriptives du langage qui s'avèrent être tout à fait suffisantes. L'interface de consultation a nécessité, quant à elle, le développement de nombreux scripts en Java pour prendre en charge des interactions plus complexes. Nous nous sommes alors très vite aperçus que ce modèle VRML+Java ne répondait absolument pas à notre besoin de prototypage rapide. VRML perd en effet toutes ses qualités de langage de description. Les non programmeurs et les générateurs de scènes (modeleurs ou autres) ne peuvent alors pas accéder à ces interactions programmées. Bien que ne satisfaisant pas nos exigences, cela nous a toutefois permis de mettre en évidence le! s carences de VRML au niveau de la description des interactions. Nos propositions d'extensions du langage découlent directement de cette deuxième expérimentation.

### Bureau 3D

La deuxième application, développée suite à l'étude des technologies 3D, est le prototype de bureau 3D ou atelier 3D comme cela est plus communément appelé. Le but d'un tel projet est de pouvoir faire coexister les applications 2D et les applications 3D. Ce besoin est né de la multiplication du nombre d'applications exploitant une interface 3D et donc de la multiplication du nombre de fenêtres 2D ouvertes pour en effectuer le rendu. Le but est de proposer un environnement 3D unique, capable de prendre en charge les deux types d'applications. Cela évite de devoir continuellement changer de métaphores de navigation et d'interaction en fonction du contenu 2D ou 3D des fenêtres.

Comme la plupart des projets équivalents, notre prototype se présente sous la forme d'une application autonome qui récupère les images des fenêtres ouvertes dans le véritable bureau 2D. Ces images sont ensuite utilisées comme des textures et plaquées sur des faces 3D que l'on peut manipuler librement dans l'espace. La manière communément employée pour récupérer les images est d'utiliser VNC d'AT&T [Richardson98] dont le but premier est d'afficher, sur un poste client, un environnement distant et de pouvoir interagir avec. Notre application fonctionne sur le même principe en utilisant directement les structures et fonctions de la Xlib.

Les interactions présentes dans notre prototype sont, pour l'instant, relativement limitées. Il est possible d'interagir par l'intermédiaire de la souris avec le contenu des fenêtres en 3D. Un changement du contenu provoqué par un clic de souris est immédiatement reflété par le moteur 3D. Comme pour tout objet 3D, il est également autorisé de déplacer et d'orienter librement les fenêtres dans l'espace. Les objets 3D ne peuvent pas former pour l'instant l'interface d'une application 3D. Ils sont chargés depuis un fichier VRML et évoluent de manière autonome.

## Extension de VRML

Un autre constat de cette thèse, est la mise en évidence des insuffisances de VRML pour la description des interactions et des animations. Le système à base d'événements et de scripts nous parait en effet compliqué et peu fiable (car non-déterministe). Nous avons proposé une extension au langage VRML pour décrire les comportements des éléments d'une scène d'une manière aussi simple et efficace que sont décrits les propriétés graphiques. Les mécanismes classiques d'interaction offerts par VRML ne conviennent pas aux non programmeurs. Les extensions proposées visent à fournir un minimum d'outils pour ces non experts afin qu'ils puissent créer des scènes réactives. Les scripts restent cependant nécessaires pour effectuer des opérations non spécifiques à la 3D.

Plusieurs organismes ou entreprises ont proposé de nouveaux nœuds pour étendre les possibilités de VRML. Parmi ceux-ci, les initiatives du consortium Web3D et des sociétés Shout3D et ParallelGraphics sont les plus importantes. Toutes les extensions proposées par ces sociétés touchent principalement aux aspects graphiques et non à l'interactivité. On note par exemple, les nœuds *Geospatial* (pour mettre en place un SIG 3D) et *Humanoid* (pour construire des avatars) ajoutés dans le format X3D, le successeur de VRML97. D'autres exemples sont les nœuds gérant le *bump mapping* ou les systèmes de particules dans le navigateur Shout3D, ou encore, le nœud gérant les textures vidéo "streamées" dans Cortona. La seule réelle extension interactive est l'adjonction d'un nœud pour gérer les interactions au clavier dans le *plugin* Cortona de ParallelGraphics. Ces différents exemples illustrent le fait que la principale préoccupation est ! de parfaire le réalisme visuel, en laissant en arrière plan l'aspect fondamental de la 3D temps réel qu'est l'interactivité.

Nous avons donc proposé d'intégrer la gestion des animations et des comportements dans la description de la scène VRML par l'adjonction de nouveaux nœuds. Et bien que cette approche soit présentée dans le cadre particulier d'une interface 3D d'accès à une bibliothèque numérisée, elle est cependant généralisable, car elle se base sur des comportements génériques. La syntaxe des fichiers n'est que légèrement modifiée et devient consistante :

utilisation du graphe de scène pour décrire les objets graphiques et leurs comportements. Ce langage VRML ainsi étendu pourra être utilisé pour le prototypage d'applications 3D, dans l'esprit des nombreux outils disponibles pour les environnements "plats".

Toujours pour étendre les possibilités de VRML, nous avons également ajouté de nouveaux nœuds pour mieux gérer la spatialisation sonore. Cette initiative est née à la fois de notre volonté de faire évoluer VRML et d'une étude sur les formats de fichier sonore utilisable dans le cadre d'un projet artistique. Dans ce projet, le but est de soumettre à des auditeurs des sons spatialisés en fonction de l'endroit où ils se trouvent. Après avoir étudié les différentes alternatives, nous nous sommes tournés vers le langage VRML. Or, dans la norme VRML actuelle, il nous est uniquement permis de positionner un son dans l'espace, ce qui n'est pas suffisant. Pour rejoindre nos attentes, nous y avons donc adjoint la possibilité de définir des effets de salle et de spécifier des filtres sonores pour gérer les occlusions. On obtient ainsi un format de fichier plus simple que celui de MPEG4 pour des possibilités quasi-identiques. L'implémentation de ces extensions sonores a été réalisée ! dans un de nos deux navigateurs VRML à l'aide de la librairie OpenAL.

## Travaux futurs

De nouveaux outils, comme Virtools Dev ou Cult3D, qui gèrent des comportements pour créer graphiquement des applications 3D, sont apparus pendant le déroulement de mon travail de thèse. Le point véritablement important est l'apparition des comportements, les interfaces graphiques pour les assembler étant plutôt restrictive. Pour cette raison, nous pensons que VRML est toujours une alternative intéressante à ces technologies propriétaires car son format est public (norme ISO), réutilisable et facilement générable. Ces deux technologies nous montrent pourtant que l'utilisation de comportements simplifiant la création d'applications 3D est d'actualité.

Tout en continuant nos études sur les bibliothèques numériques et le bureau 3D, nous avons l'intention de poursuivre l'étude des comportements avec VRML. La description des scènes au format MPEG4 étant très proche du format VRML, cela nous assure de pouvoir réutiliser nos extensions sous la forme de *codecs* spécifiques, dès lors que MPEG4 se sera imposé comme le "méta-standard" que l'on nous annonce depuis quatre ans.

### Comportements

La spécification des comportements pour VRML n'en est qu'à ses débuts. Nous devons maintenant augmenter leur nombre pour obtenir une panoplie plus complète. La manière la plus naturelle pour cela, est d'étudier de nouvelles applications. Nous pensons, en particulier, aux nouvelles fonctionnalités du trépied qui effectueront de nouvelles opérations mais aussi de toutes nouvelles applications que nous envisageons d'étudier comme les interfaces 3D d'accès à la télévision numérique.

Nous apporterons une attention toute particulière à un point non encore résolu : la dynamicité des scènes. Les navigateurs VRML s'appuyant sur le protocole HTTP, il n'est pas naturel d'effectuer des communications synchrone avec le serveur. Deux mécanismes

peuvent cependant être mis en place pour simuler ce dialogue, tous deux à l'initiative du client mais exécutés sur le serveur comme des scripts CGI. Le premier est de contacter le serveur sur une action de l'utilisateur. Plus généralement, cela suppose que des actions possibles dans un nœuds de comportements peuvent être routés vers un élément externe à la scène (le serveur par exemple). Dans le cas où notre format VRML augmenté est utilisé comme une ressource dans un programme 3D, les actions devront pouvoir être routées vers un autre objet 3D indépendant ou récupérées par l'application.

Le deuxième mécanisme à mettre en place est la mise à jour périodique de la scène par appel au serveur. Ceci pourra être fait avec une contrainte temporelle forte pour les systèmes temps réel. Dans ce cas, les communications entre clients et serveur seront privilégiées par rapport à l'affichage des scènes. La deuxième méthode est de privilégier l'affichage et de contacter le serveur dès que celui-ci le permettra. Cette méthode sera utilisée lorsque l'affichage dépend peu des réponses du serveur (dans le cas d'un jeu en réseau par exemple).

### Gestion de divers dispositifs de pointage

Les nœuds de comportement que nous gérons pour l'instant sont sensibles à la souris. Celle-ci n'est certainement pas le meilleur dispositif de pointage pour manipuler et évoluer dans une scène 3D ; même si elle peut toutefois s'avérer pratique pour effectuer des sélections d'objets. Nous souhaiterions intégrer d'autres dispositifs de pointage. Leur prise en charge par nos nœuds de comportements n'est pas réellement un problème. Nous avons utilisé des termes génériques pour spécifier les interactions qui s'adaptent *a priori* à tous les dispositifs de pointage. Le principal problème est d'intégrer la gestion des autres périphériques dans notre navigateur VRML. Pour cela, nous souhaiterions disposer d'une nouvelle couche d'abstraction matérielle, schématisée dans la figure 6.2, que nous nommons la DHAL (*Device Hardware Abstraction Layer*). Alors que la couche d'accélération graphique HAL (*Hardware Abstraction Layer*) prend en charge de ! manière transparente les fonctions câblées sur les cartes graphiques, la DHAL fournirait aux programmeurs une série de fonctions pour déterminer quels dispositifs sont présents, quels sont leurs caractéristiques et pour donner les moyens d'y accéder. Une telle interface entre les applications et les pilotes des périphériques serait écrite une seule et accéssible à volonté.

La DHAL devrait être implémentée et utilisée de la même manière que la couche HAL. Par son intermédiaire, une application 3D pourra choisir quels périphériques sont les mieux adaptés aux diverses tâches. Par exemple, si un dispositif de pointage 3D isométrique, une souris et un *joystick* à retour de forces sont disponibles, l'application récupère en premier toutes leurs caractéristiques. Elle pourra ensuite déterminer que le périphérique 3D est mieux adapté à la manipulation des objets et les deux périphériques 2D pour la sélection et la navigation. Une telle interface entre une application et les périphériques d'entrée apportera une souplesse pour la gestion des interactions non accessible jusqu'alors. Le moyen actuel est de prendre en charge les dispositifs par programmation explicite dans l'application. La DHAL permettra de déléguer cette fastidieuse programmation.

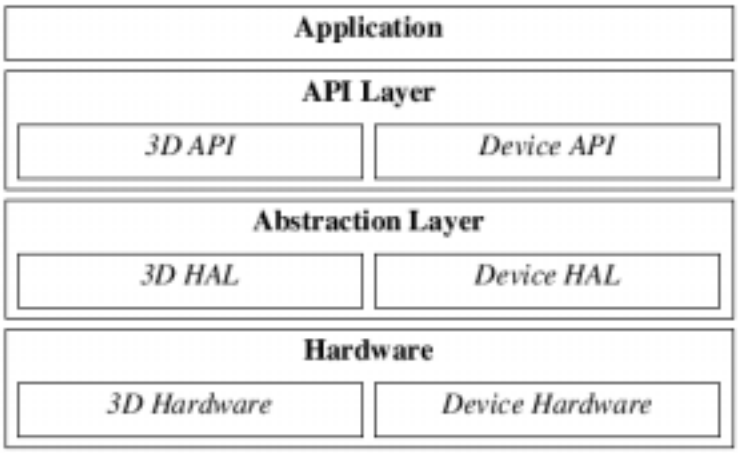


**Fig. 6.2** La couche d'abstraction matérielle DHAL.

### Interfaces pour les bibliothèques numériques

L'apport des heuristiques visuelles que nous avons mises en place n'est accessible que lorsque les dos des livres ont été numérisés. Or, pour la plupart des bibliothèques numériques, ces informations n'ont pas été acquises lors de la conception. On peut également penser que dans bien des cas, la couverture ne sera pas assez discriminante. A partir des quelques informations disponibles sur la constitution physique des ouvrages (pagination, date de publication, ...), il serait peut-être possible de bâtir une interface plus métaphorique, reposant sur une "grammaire" de formes préétablies dont il faudrait dresser l'inventaire.

Tous les outils et les fonctionnalités imaginés ne sont pas encore accessibles par le trépied. Il faudra notamment mener une réflexion sur de nouveaux outils de navigation dans la structure même des documents et les intégrer dans notre application. Lors de la mise en place du CNUM, nous avons pu constater que celle-ci était parfois d'une grande complexité pour les textes scientifiques anciens (rôle des illustrations, des annexes, ...) et les normes de description de corpus (TEI et CES) ne traitent pas complètement de ces aspects. Il conviendrait donc d'abord d'étendre ces dernières.

Nos métaphores pour basculer de l'interface de recherche à l'interface de lecture n'ont pas non plus encore été implémentées. Il en est de même pour la troisième activité : l'annotation et le partage d'informations entre utilisateurs. Pour cela, nous pourrions profiter de nos futures extensions pour communiquer avec le serveur qui prendrait en charge la synchronisation entre plusieurs lecteurs. Ces différents aspects non encore inclus dans notre application permettront d'extraire de nouveaux comportements que nous intégrerons à l'ensemble déjà disponible.

Une solution envisagée pour étendre notre bibliothèque 3D est de l'inclure à l'intérieur de notre environnement 3D. Le but de notre bureau 3D est de mêler à terme les contenus 2D et 3D locaux ou distants. La bibliothèque se comportera comme un contenu 3D distant exécuté en local. Dans ce contexte, tout traitement de texte pourra servir à prendre des notes et

les navigateurs HTML à élargir les recherches. De plus, nous envisageons d'étudier la possibilité de partager les applications entre plusieurs utilisateurs à la manière des logiciels de visio-conférences tels que Netmeeting.

Une question importante concernant la validation de nos expérimentations par des tests utilisateurs n'a pas été soulevée. L'arrivée prochaine d'une nouvelle campagne de numérisation des articles de la Nature offrira sur le CNUM, un corpus conséquent et plus diversifié. Il s'accompagnera nécessairement d'un plus grand nombre de visiteurs avec lesquels nous pouvons obtenir de précieux résultats. De plus, dans un avenir proche nous proposerons les textes de la bibliothèque ABU au format image. Là aussi, nous pouvons soumettre nos interfaces 3D à de nombreux lecteurs aux connaissances informatiques hétéroclites.

Pour valider nos interfaces, mais également obtenir un retour sur nos productions, nous pensons utiliser différentes méthodes. La première, se fera sous la forme d'un questionnaire pour les utilisateurs qui auront essayé nos interfaces. La première version, en VRML, est présente depuis environ un an sur le site CNUM (`http ://cnum.cnam.fr/vrml`). En y adjoignant un formulaire pour collecter les remarques et critiques des utilisateurs, nous pourrions faire progresser notre prototype.

La deuxième méthode envisagée est d'intégrer à notre prototype des mesures d'efficacité prise en fonction des déplacements et des manipulations des utilisateurs. Ces *logs* 3D nous donneront des informations telles que le temps moyen passé à utiliser notre bibliothèque, le temps mis à trouver un ouvrage (en supposant que l'ouvrage ouvert après une requête est l'ouvrage recherché). Ces résultats pourront alors être comparés avec les *logs* actuels du serveur CNUM.

Enfin, nous pourrions effectuer des tests contrôlés, tels que ceux faits par [Cockburn01], dans lesquels nous demanderions aux utilisateurs de rechercher, sélectionner et lire un ouvrage avec les interfaces 2D et 3D. Les temps relevés seront alors complètement comparables et les remarques directes et moins ambiguës.

### Bureau 3D

Pour le bureau 3D, nous envisageons la création d'une extension pour les serveurs X afin de gérer le problème de récupération des textures des fenêtres. Le but de cette extension sera uniquement de renvoyer les images des fenêtres sans devoir les positionner dans un *frame buffer*. Cette simple extension devrait nous permettre alors de transformer notre prototype en gestionnaire de fenêtre XWindow. Pour cela, nous devrons proposer un nouvel habillage et de nouvelles interactions pour les fenêtres, afin que l'interface ne paraisse pas aussi dépouillée. Nous envisageons d'utiliser la même métaphore que dans la bibliothèque numérique, c'est-à-dire le trépied. Le but ultime est de faire cohabiter nos deux projets en utilisant le bureau 3D comme interface de rendu de la bibliothèque numérique et le trépied VRML comme *widgets* habillant les applications.

D'autres aspects devront avant cela être résolus. Les premiers concernent les problèmes de rendu des textes pour obtenir une meilleur lisibilité. Des algorithmes d'antialiasing des géométries et des textures devront être utilisés. D'autres interactions avec les applications devront être mises en place. En particulier, les interactions au clavier qui sont des plus

importantes. Les intégrer ne devrait pas poser de problèmes puisque le mécanisme d'attribution des événements à une fenêtre est le même quelque soit le type d'événement. Les interactions permettant de manipuler les élément constituant l'interface (applications ou objets 3D) devront également être considérées. Pour cela, nous bénéficierons du développement de notre couche d'abstraction DHAL.

Le bureau, étendu par l'adjonction d'une extension 3D au serveur XWindow, par l'accès à une couche DHAL et par la disponibilité de ressources décrites grâce à notre langage VRML augmenté (figure 6.3), nous fournira un environnement complet pour le développement de nouvelles applications. Nous pensons, entre autres, à une interface pour accéder aux émissions des télévisions numériques. Le bureau 3D deviendrait alors une interface unique pour accéder à divers types de documents numériques : les livres, les applications, les films et émissions.

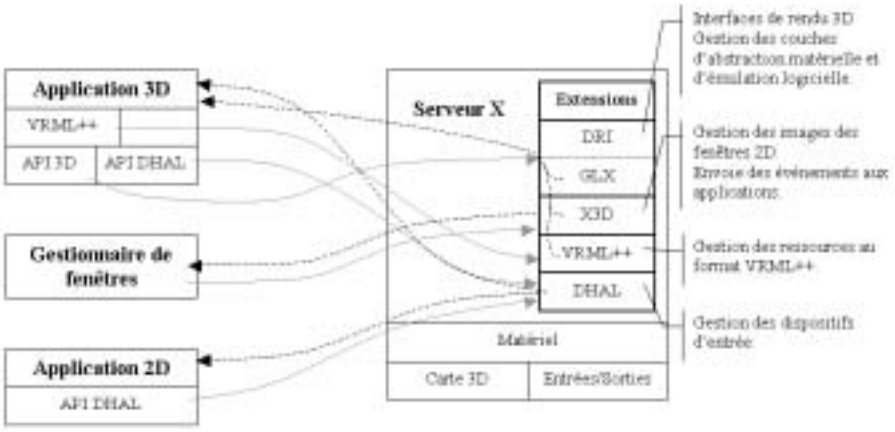


**Fig. 6.3** L'architecture pressentie de la deuxième version du bureau 3D.

# *Publications*


[1] P. Cubaud, C. Thiria, A. Topol. "Experimenting a 3D Interface for the Access to a Digital Library", *In Proceedings of the ACM Digital Lybraries Conference (DL'98)*, Pittsburgh, PA, USA, June 23-26 1998.

[2] P. Cubaud, A. Topol. "A WWW-based digital library for antiquarian collections", *Rapport technique CEDRIC No 208*, 1999.

[3] A. Topol. "Immersion of XWindow applications into a 3D workbench", *In Proceedings of the ACM Computer-Human Interaction Conference (CHI'2000)*, The Hague, Netherlands, April 2000.

[4] P. Cubaud, A. Topol, D. Vodislav. "Les limites de VRML pour les comportements interactifs : étude de cas", *Actes de la Conférence Ergonomie et Interaction Homme-Machine (ERGO-IHM'2000)*, Biarritz, France, Octobre 2000.

[5] P. Cubaud, A. Topol, L. Bacon. "L'utilisation des techniques graphiques avancées pour le contrôle commande et la supervision", *Rapport technique pour la division Recherche et Développement EDF Pôle Industrie*, Décembre 2000.

[6] P. Cubaud, A. Topol. "A VRML-based user interface for an online digitalized antiquarian collection", *In Proceedings of the ACM Web3D'2001 Conference*, Paderdorn, Germany, February 2001.

[7] A. Topol. "VRML : étude, mise en oeuvre et applications", *Mémoire d'ingénieur CNAM*, Juillet 2001.

[8] A. Topol. "Description des comportements dans une scène 3D", *Atelier sur les interfaces 3D, conférence IHM-HCI 2001*, Lille, France, Septembre 2001.

[9] S. Natkin, F. Schaeffer, A. Topol. "Functional Specification of a Distributed and Mobile Architecture for Virtual Sound Space Systems", *In Proceedings of the International Computer Music Conference (ICMC'2001)*, La Havana, Cuba, September 2001.

[10] A. Topol, F. Schaeffer. "Enhancing sound description in VRML", *In Proceedings of the International Computer Music Conference (ICMC'2001), La Havana, Cuba, September 2001.*

[11] P. Cubaud, P. Stokowski, A. Topol. "3D Metaphors to access a digitalized library", *In Proceedings of the Virtual Reality International Conference (VRIC'2002)*, Laval, France, June 2002.

[12] P. Cubaud, P. Stokowski, A. Topol. "Binding Browsing and Reading Activities in a 3D Digital Library", *In Proceedings of the ACM/IEEE Joint Conference on Digital Libraries (JCDL'2002)*, Portland, Oregon, USA, July 2002.

# *Références*


[Abisror02] J.-M. Abisror. Vrml97 sous java-opengl. Mémoire d'ingénieur, Conservatoire National des Arts et Métiers, June 2002.

[André95] J. André, J.-D. Fekete, and H. Richy. Traitement mixte image/texte de documents anciens. In *congrès Gutenberg*, pages 75–85, 1995.

[Andrews95] K. Andrews. Visualizing cyberspace: Information visualisation in the harmony internet browser. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis'95)*, pages 97–104. IEEE CS Press, 1995.

[Avaro97] O. Avaro. The mpeg-4 system and description languages : A way ahead in audio visual information representation. *Signal Processing: Image Communication, Special Issue on MPEG-4*, 9(4):385–431, May 1997.

[Baecker95] R. Baecker, J. Grudin, W. Buxton, and S. Greenberg. *Readings in Human-Computer Interaction : Toward the Year 2000.* Morgan-Kauffmann, 2nd edition, 1995.

[Balaguer95] J.-F. Balaguer and E. Gobbetti. i3d : A high-speed 3d web browser. In *VRML*, pages 69–76, 1995.

[Balaguer99] J.-F. Balaguer. Less is more : The power of simplicity. In *Proceedings of the ACM VRML'99 Conference.* ACM Press, February 1999.

[Bannon83] L. Bannon, A. Cypher, S. Greenpan, and M.L. Monty. Evaluation and analysis of user's activity organization. In *Proceedings of the ACM SIGCHI'83 Conference on Human Factors in Computing Systems*, pages 54–57. ACM Press, 1983.

[Battista94] G. Battista, P. Eades, R. Tamassia, and I.G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry: Theory and Applications*, 4(5):235–282, 1994.

[Battista99] G. Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Algorithms for the Visualisation of Graphs.* Prentice Hall, 1999.

[Bederson94] B. B. Bederson. Pad++ : A zoomable graphical interface. In *Proceedings of the ACM SIGCHI'94 Conference on Human Factors in Computing Systems*, Boston, MA, USA, April 1994. ACM Press.

[Bederson96] B. B. Bederson, J. Hollan, J. Stewart, D. Rogers, A. Druin, D. Profit, M. Ijadi, K. Perlin, and J. Meyer. A zooming web browser demonstration. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Seattle, Washington, USA, November 1996. ACM Press.

[Bederson00] B. B. Bederson. Fisheye menus. In *Proceedings of the User Interface Software and Technology (UIST) 2000 Conference*, pages 217–225, 2000.

[Benedikt91] M. Benedikt. Cyberspace: Some proposals. In *Cyberspace: First Steps*, pages 273–302. MIT Press, 1991.

[Benford93] S. Benford and L. E. Fahlen. A spatial model of interaction in large virtual environments. In *ECSCW*, pages 107–118, 1993.

[Benford94a] S. Benford. Experience of using 3d graphics in database visualisation. Research report, Computing Department, Lancaster University, October 1994.

[Benford94b] S. Benford and J. Mariani. Populated information terrains: Virtual environments for sharing data. Research report cscw/4/1994, Computing Department, Lancaster University, October 1994.

[Benford95] S. Benford, D. Snowdon, C. Greenhalgh, R. Ingram, I. Knox, and C. Brown. Vr-vibe: A virtual environment for co-operative information retrieval. In *Proceedings of Eurographics'95*, pages 349–360, Maastricht, Netherlands, August 1995.

[Benford96] S. Benford, C. Brown, G. Reynard, and D. Snowdon. The internet foyer. In *Proceedings of the Third Conference of the UK VRSIG*, DeMontford University, England, UK, July 1996.

[Benford97] S. Benford, D. Snowdon, C. Brown, and G. Reynard. Visualising and populating the Web: Collaborative virtual environments for browsing, searching and inhabiting Webspace. *Computer Networks and ISDN Systems*, 29(15):1751–1761, 1997.

[Bernstein91] M. Bernstein, P.J. Brown, M. Frisse, R. Glushhko, G. Landow, and P. Zellweger. Structure, navigation, and hypertext: The status of the navigation problem. In *Proceedings of the ACM Conference on Hypertext and Hypermedia (HT'91)*, pages 363–366, 1991.

[Bibes97] M. Bibes and E. Fléty. Classification des capteurs. Pédagogie, IRCAM, November 1997.

[Bier93] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses : The see-through interface. In *Proceedings of the Computer Graphics Annual Conference*, pages 73–80, Anaheim, California, USA, August 1993. ACM Press.

[Bly86] L. Bly and J. Rosenberg. A comparison of tiled and overlapping windows. In *Proceedings of the ACM SIGCHI'86 Conference on Human Factors in Computing Systems*, pages 101–106, New York, NY, USA, August 1986. ACM Press.

[Boyle93] J. Boyle, S. Leishman, J. Fothergill, and P. Gray. Development of a visual query language. Research report, Aberdeen University, 1993.

[Boyle96] J. Boyle, S. Leishman, and P. M. D. Gray. From wimp to 3d: the development of amaze. *Journal of Visual Languages and Computing*, 7:291–319, 1996.

[Brooks83] R. Brooks. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, 18(6):543–554, 1983.

[Brown93] M.H. Brown and M.A. Najork. Algorithm animation using 3d interactive graphics. Research report, DIGITAL, Systems Research Center, September 1993.

[Burdea93] G. Burdea and Ph. Coiffet. *Réalité virtuelle*. Hermès, 1993.

[Buxton86] W. Buxton and B. A. Myers. A study in two-handed input. In *Proceedings of SIGCHI'86. Conference on Human Factors in Computing Systems*, pages 321–326, Boston, MA, USA, April 1986. ACM Press.

[Callan92] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *Proceedings of DEXA-92, 3rd International Conference on Database and Expert Systems Applications*, pages 78–83, 1992.

[Card87] S.K. Card and A.H. Henderson Jr. A multiple virtual workspace interface to support user task switching. In *Proceedings of the CHI+GI 1987 Conference*, pages 53–59. ACM Press, April 1987.

[Card91] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The information visualizer, an information workspace. In *Proceedings of SIGCHI'91 Conference on Human Factors in Computing Systems*, pages 181–188, New Orleans, Louisiana, USA, April 1991. ACM Press.

[Card96] S. Card, G. Robertson, and W. York. The webbook and the web forager : An information workspace for the world-wide-web. In *Proceedings of ACM SIGCHI'96 Conference on Human Factors in Computing Systems*, Vancouver, Canada, April 1996. ACM Press.

[Carlsson93a] C. Carlsson and O. Hagsand. The distributed interactive virtual environments – architecture and applications. In *Proceedings of Computing and Control Division Colloquium on Distributed Virtual Reality*, London, England, May 1993.

[Carlsson93b] C. Carlsson and O. Hagsand. Dive - a multi-user virtual reality system. In *Proceedings of VRAIS'93, IEEE Virtual Reality Annual International Symposium*, pages 394–400, 1993.

[Celentano99] A. Celentano. Structuring and presenting www documents in a vr metaphore. In *5ème conférence internationale Hypertextes, Hypermédias & Internet : Réalisations, Outils & Méthodes*, Université Paris VIII, Paris, France, September 1999.

[Chalmers92] M. Chalmers and P. Chitson. Bead: Explorations in information visualisation. In *Proceedings of SIGIR'92*. ACM Press, June 1992. published as a special issue of SIGIR forum.

[Chalmers95] M. Chalmers. Design perspectives in visualising complex information. In *Proceedings of IFIP 3rd Visual Databases Conference*, Lausanne, Switzerland, March 1995.

[Chen88] M. Chen. A study in interactive 3d rotation using 2d control devices. *ACM Computer Graphics*, 22(4):121–127, August 1988.

[Chen99] C. Chen. *Information Visualisation and Virtuals Environments*. Springer, 1999.

[Cleary99] D. Cleary and D. O'Donoghue. Visualexpresso: Generating a virtual reality internet. In *P. Sloot, M. Bubak, A. Hoekstra, and B. Hertzberger (Ed.) High Performance Computing and Networking HPCN-99: Virtual Reality Workshop*, pages 797–806. Springer-Verlag, 1999.

[Cleary00] D. Cleary and D. O'Donoghue. Generating a topically focused virtual-reality internet. In *Proceedings of INET 2000 - The 10th Annual Conference of The Internet Society*, pages 18–21, Yokohama, Japan, July 2000.

[Cockburn01] A. Cockburn and B. McKenzie. 3d or not 3d? evaluating the effect of the third dimension in a document management system. In *Proceedings of the ACM SIGCHI'2001 Conference on Human Factors in Computing Systems*, Seattle, WA, USA, March 2001.

[Colebourne94] A. Colebourne. Populated information terrains: supporting the cooperative browsing of on-line information. Research report cscw/13/1994, Centre for Research in CSCW, Lancaster University, 1994.

[Conner92] D. Conner, S. Snibbe, K. Herndon, D. Robbins, R. Zeleznik, and A. van Dam. Three-dimensional widgets. *ACM SIGGRAPH'1992 Symposium on Interactive 3D Graphics*, 25(2):183–188, March 1992.

[Conway94] M. Conway, R. Pausch, R. Gossweiler, and T. Burnette. Alice : A rapid prototyping system for building virtual environments. In *Proceedings of SIGCHI'94 Conference on Human Factors in Computing Systems*, pages 295–296, Boston, MA, USA, April 1994. ACM Press.

[Conway00] M. Conway, R. Pausch, R. Gossweiler, and T. Burnette. Lessons learned from building a 3d system for novices. In *Proceedings of SIGCHI'2000 Conference on Human Factors in Computing Systems*, The Hague, Netherlands, April 2000. ACM Press.

[Cox95] K. C. Cox and S. G. Eick. 3d displays of internet traffic. In *Proceedings of Information Visualization '95 Symposium (InfoVis '95)*, pages 129–131. IEEE CS Press, 1995.

[Cubaud98] P. Cubaud, C. Thiria, and A. Topol. Experimenting a 3d interface for the access to a digital library. In *Proceedings of the third conference on Digital Libraries*, Pittsburg, USA, June 1998. ACM Press.

[Cubaud00] P. Cubaud, A. Topol, and D. Vodislav. Les limites de vrml pour les comportements interactifs : étude de cas. In *Actes de la conférences ERGOIHM'2000*, Biarritz, France, October 2000.

[Cubaud01] P. Cubaud and A. Topol. A vrml-based user interface for an online digitalized antiquarian collection. In *Proceedings of the ACM Web3D'2001 Conference*, Paderdorn, Germany, February 2001. ACM Press.

[Cubaud02a] P. Cubaud, P. Stokowski, and A. Topol. 3d metaphors to access a digitalized library. In *Proceedings of the Virtual Reality International Conference (VRIC'2002)*, Laval, France, June 2002.

[Cubaud02b] P. Cubaud, P. Stokowski, and A. Topol. Binding browsing and reading activities in a 3d digital library. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries (JCDL'2002)*, Portland, Oregon, USA, July 2002.

[Czerwinski99] M.P. Czerwinski, M. van Dantzich, G. Robertson, and H. Hoffman. The contribution of thumbnail image, mouse-over text and spatial location memory to web page retrieval in 3d. In *Proceedings of INTERACT'99*, pages 163–170, 1999.

[Darken96] R. Darken and J.L. Sibert. Navigating large virtual spaces. *International Journal of Human-Computer Interaction*, 8:49–62, 1996.

[Dechelle98] F. Dechelle, R. Borghesi, M. De Cecco, E. Maggi, B. Rovan, and N. Schnell. jmax: a new java-based editing and control system for real-time musical applications. In *Proceedings of the International Computer Music Conference (ICMC'98)*, October 1998.

[Diehl00] S. Diehl and J. Keller. Vrml with constraints. In *Proceedings of the Web3D/VRML'2000 Conference*, Monterey, CA, USA, February 2000. ACM Press.

[Doornbos00] H. Doornbos. Le mmx d'intel et le 3dnow! d'amd : comparaison des instructions et des parties spécifiques des processeurs des deux fabricants. Rapport d'oral probatoire, Conservatoire National des Arts et Métiers, January 2000.

[Dumas99a] C. Dumas, S. Degrange, G. Saugis, C. Chaillou, M.-L. Viaud, and P. Plénacoste. Spin : a 3-d interface for cooperative work. *Virtual Reality Society Journal*, 1999.

[DUMAS99B] Cédric Dumas. *Un modèle d'interaction 3D : Interactions homme-machine et homme-machine-homme dans les interfaces 3D pour le TCAO synchrone*. Université des Sciences et Technologies de Lille, October 1999.

[EADES84] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[EMMERIK90] M. Emmerik. A direct manipulation technique for specifying 3d object transformations with a 2d input device. *Journal of the Yugoslav Commitee ETAN*, 31(1–2):95–99, 1990.

[FAIRCHILD88] K.M. Fairchild, S.E. Poltrock, and G.W. Furnas. Semnet: Three-dimentional graphic representation of large knowledge bases. In *Cognitive Science and its Applications for Human-Computer Interaction*. Lawrence Erlbaum Associates, 1988.

[FAIRCHILD93] K.M. Fairchild. Information management using virtual reality-based visualizations. In Academic Press Professional, editor, *Virtual Reality: Applications and Explorations*, pages 45–74, 1993.

[FLANAGAN94] D. Flanagan. *X Toolkit Intrinsics Reference Manual*. O'Reilly, 2nd edition, February 1994.

[FLANAGAN96] D. Flanagan. *Java in a nutshell*. O'Reilly, 1996.

[FLÉTY97] E. Fléty. Sonars à ultrason. Rapport de stage, IRCAM, 1997.

[FOLEY90] J.D. Foley, A. Van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics - principes and practice*. Addison-Wesley, New York, 1990.

[FOWLER96] R.H. Fowler, A. Kumar, and J.L. Williams. Visualizing and browsing www semantic content. In *Proceedings of the First Conference on Emerging Technologies and Applications in Communications*, Portland, Oregon, USA, May 1996.

[FOX93] E. Fox. Users, user interfaces, and objects: Envisioj, a digital library. *Journal of the American Society for Information Science*, 44(8), 1993.

[FOX94] E. Fox. Improving education through a computer science digital libray with three types of www servers. Technical report, Technical Repport of Dept. of Computer Science, Virginia Tec, Blacksburg , VA 24061-0106, October 1994.

[FOX97] E. Fox, J. Eaton, G. McMillan, N. Kipp, P. Mather, T. McGonigle, W. Schweiker, and B. DeVane. Networked digital library ot theses and disertations: An international effort unlocking university resources. *D-lib magazine*, September 1997.

[FRUCHTERMAN91] T.M.J. Fruchterman and E.M. Reingold. Graph drawing by force-directed placement. *Software Practice and Experience*, 2(11), November 1991.

[Fuchs99] P. Fuchs. Les interfaces de la réalité virtuelle. In *Les journées de Montpellier*, 1999.

[Fuchs02] P. Fuchs. Les interfaces de la réalité virtuelle. In *Proceedings of the Virtual Reality International Conference 2002*, June 2002.

[Funkhouser98] T. Funkhouser, I. Carlbom, G. Elko, G. Pingali, M. Sondhi, and J. West. A beam tracing approach to acoustic modeling for interactive virtual environments. *Computer Graphics*, 32(Annual Conference Series):21–32, 1998.

[Furnas86] G. W. Furnas. Generalized fisheye views. In *Proceedings of SIGCHI'86. Conference on Human Factors in Computing Systems*, pages 16–34, Boston, MA, USA, April 1986. ACM Press.

[Furnas95] G. W. Furnas and B. B. Bederson. Space-scale diagrams: Understanding multiscale interfaces. In *Proceedings of SIGCHI'95. Conference on Human Factors in Computing Systems*, Denver, CO, USA, May 1995. ACM Press.

[Gallic99] J.-M. Le Gallic. La visualisation des données géographiques – l'approche vrml – java. Mémoire d'ingénieur, Conservatoire National des Arts et Métiers, June 1999.

[Gelernter92] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, February 1992.

[Gershon95a] N. Gershon and S. G. Eick. Visualization's new task : Making sense of information. *IEEE Spectrum*, pages 38–56, November 1995.

[Gershon95b] N. Gershon, J. Levasseur, J. Winstead, J. Croall, A. Pernick, and W. Ruh. Case study: Visualizing internet resources. In IEEE Press, editor, *Proceedings of the Information Visualisation Symposium 1995*, pages 122–128, Los Alamitos, CA, USA, October 1995.

[Gossweiler93] R. Gossweiler, C. Long, S. Koga, and R. Pausch. Diver: A distributed virtual environment research platform. *IEEE Symposium on Research Frontiers in Virtual Reality*, pages 38–56, October 1993.

[Gould84] J.D. Gould and N. Grischokowsky. Doing the same work with hard copy and with cathode-ray tube (crt) computers terminals. *Human Factors*, 26(3):323–337, 1984.

[Gould87] J.D. Gould, L. Alfaro, R. Fonn, R. Haupt, A. Minuto, and J. Salaun. Why reading was slower from crt displays than from paper. In *Proceedings of ACM CHI+GI 87*, pages 7–11, April 1987.

[Gray90] P.D. Gray, K.W. Waite, and S.W. Draper. Do-it yourself iconic displays, humancomputer interaction. In *INTERACT '90: Proceedings of the IFIP TC 13 Third International Conference on Human-Computer Interaction*, pages 639–644, 1990.

[Harel88] D. Harel. On visual formalisms. *Communications of the ACM*, 31(5), May 1988.

[Hearst96] M. Hearst, G. Kopec, and D. Brotsky. Research in support of digital libraries at xerox parc. part ii. June 1996.

[Hemmje93] M. Hemmje. A 3d based user interface for information retrieval systems. In *Proceedings of IEEE Visualization '93, Workshop on Database Issues for Data Visualization*, October 1993.

[Hemmje94] M. Hemmje, C. Kunkel, and A. Willet. Lyberworld - a visualization user interface supporting fulltext retrieval. In *Proceedings of ACM SIGIR '94*, July 1994.

[Henderson86] D.A. Henderson and S.K. Card. Rooms: The use of multiple virtual workspaces to reduce spatial contention in a window-based graphical user interface. *ACM Transactions on Graphics*, 5(3), July 1986.

[Hendley95a] R. Hendley and N. Drew. Visualisation of complex systems. In *Proceedings of the HCI'95 Conference*, 1995.

[Hendley95b] R. J. Hendley, N. S. Drew, A. M. Wood, and R. Beale. Narcissus: Visualising information. In *Proceedings of the IEEE Symposium on Information Visualization, InfoVis*, pages 90–96. IEEE Computer Soc. Press, 1995.

[Herder97] J. Herder. Tools and widgets for spatial sound authoring. In *Proceedings of Compugraphics'97*, December 1997.

[Herman00] I. Herman, G. Melançon, and M. S. Marshall. Graph visualisation and navigation in information visualisation. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–42, 2000.

[Huopaniemi96] J. Huopaniemi, L. Savioja, and T. Takala. Diva: virtual audio reality system. In *Proceedings of ICAD'96*, Palo Alto, California, USA, November 1996.

[Ingram96] R. Ingram, S. Benford, and J. Bowers. Building virtual cities: applying urban planning principles to the design of virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST'96)*, pages 83–91, July 1996.

[Johnson91] B. Johnson and B. Shneiderman. Treemaps: A space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd International IEEE Visualization Conference*, pages 284–291, San Diego, CA, USA, October 1991.

[Johnson92] B. Johnson. Treeviz: Treemap visualization of hierarchically structured information. In *Proceedings of SIGCHI'92 Conference on Human Factors in Computing Systems*, pages 369–370, Monterey, CA, USA, May 1992. ACM Press.

[Jones86] W. Jones and S. Dumais. The spatial metaphor for user interfaces: Experimental tests of reference by location versus names. *ACM Transactions on Office Information Systems*, 4(1):42–63, 1986.

[Jot95] J.-M. Jot and O. Warusfel. Spat˜ : A spatial processor for musicians and sound engineers. In *Proceedings of the International Conference on Acoustics and Musical Research*, 1995.

[Jot97] J.-M. Jot. Efficient models for reverberation and sitance rendering in computer music and virtual audio reality. In *Proceedings of the 1997 International Computer Music Conference*, September 1997.

[Kandegon97] E. Kandegon and B. Shneiderman. Elastic windows: Evaluation of multi-window operations. In *Proceedings of SIGCHI'97 Conference on Human Factors in Computing Systems*, pages 250–257. ACM Press, 1997.

[Kings95] N.J. Kings. Software visualisation. Technical report, British Telecommunications, report for the Corporate Research Programme, 1995.

[Kipp97] N. Kipp. A virtual digital library interface with a spacial metaphore. Technical report, Dept. of Computer Science, Virginia Tech, December 1997.

[Koike93] H. Koike and H. Yoshihara. Fractal approaches for visualizing huge hierarchies. In *Proceedings of 1993 IEEE/CS Symposium on Visual Languages (VL'93)*, pages 55–60, Los Alamitos, California, USA, 1993. IEEE Computer Society Press.

[Koike95] H. Koike. Fractal views: a fractal-based method for controlling information display. *ACM Transactions on Information Systems*, pages 305–323, July 1995.

[Krohn95] U. Krohn. Vineta: visualizing navigational information retrieval. In *Proceedings of the Second International Workshop on Foundations of Advanced 3D Information Visualization*, Glasgow, Scotland, UK, July 1995.

[Krohn96a] U. Krohn. Vineta: Navigation through virtual information spaces. In T. Cartaci, editor, *AVI: Advanced Visual Interfaces*, Gubbio, Italy, May 1996. ACM.

[Krohn96b] U. Krohn. *Visualization for Retrieval of Scientific and Technical Information.* 1996.

[Lagus96] K. Lagus, S.K. Honkela, and T. Kohonen. Self-organizing maps of document collections: a new appraoch to interactive exploration. In *Proceedings of the Second International Conference on Knowledge Discovery & Data mining (KDD'96)*, Portland, Oregon, USA, August 1996.

[Lamping95] J. Lamping, R. Rao, and P. Pirolli. A focus + context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of*

*SIGCHI'95. Conference on Human Factors in Computing Systems*, pages 401–408, Denver, CO, USA, May 1995. ACM Press.

[Leach97] G. Leach, G. Al-Qaimari, M. Grieve, N. Jinks, and C. McKay. Elements of the three-dimensional graphical user interface. In *Proceedings of INTERACT'97*, Sydney, Australia, July 1997.

[Lesk97] M.A. Lesk. *Practical Digital Libraries (Books, Bytes and Bucks)*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1997.

[Lin91] X. Lin, D. Soergel, and G. Marchionini. A self-organizing semantic map for information retrieval. In *Proceedings of the Fourteeth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, Chicago, Illinois, USA, October 1991. ACM Press.

[Mackinlay91] J. D. Mackinlay, G. G. Robertson, and S. K. Card. The perspective wall: detail and context smoothly integrated. In *Proceedings of SIGCHI'91 Conference on Human Factors in Computing Systems*, pages 401–408, New Orleans, Louisiana, USA, April 1991. ACM Press.

[Mariani92] J.A. Mariani and R. Lougher. Triplespace: an experiment in a 3d graphical interface to a binary relational database. *Interacting with Computers: The Interdisciplinary Journal of Human-Computer Interaction*, 4(2):147–162, August 1992.

[Marin96] M. Marin. *Etude de la localisation en prise et restitution pour la téléconférence de qualité*. Université des Mines, Le Mans, October 1996.

[Marques98] A. Marques. Conception et développement d'un logiciel client serveur mpeg-4 pour des applications de communication de groupe et consultation multimédia. Mémoire d'ingénieur, Conservatoire National des Arts et Métiers, June 1998.

[Marrin98] C. Marrin. Beyond vrml. Technical report, 1998.

[Mengel98] A. Mengel. *Une histoire de la lecture*. Edition Actes Sud, Arles, France, 1998.

[Morris86] J. Morris, M. Satyanarayanan, M. Conner ans J. Howard, D. Rosenthal, and F. Smith. Andrew: a distributed personal computing environment. *Communications of the ACM*, 29(3):184–201, March 1986.

[Mukherjea95] S. Mukherjea and J. Foley. Visualizing the world-wide web with the navigational view builder, technical report 95-09. Technical report, Graphics, Visualisation and Usability Center, Georgia Tech, USA, 1995.

[Munzner95] T. Munzner and P. Burchard. Visualizing the structure of the World Wide Web in 3D hyperbolic space. In *Proceedings of the VRML 1995 Symposium*, pages 33–38. ACM Press, 1995.

[Munzner98] Tamara Munzner. Drawing large graphs with H3Viewer and Site Manager. In *Graph Drawing*, pages 384–393, 1998.

[Myers88] B.A. Myers. Window interfaces: A taxonomy of window manager user interfaces. *IEEE Computer Graphics and Applications*, 8(5):65–84, September 1988.

[Myers90] B.A. Myers. Taxonomies of visual programming and program visualisation. *Journal of Visual Languages and Computing*, 1:97–123, 1990.

[Natkin98] S. Natkin. Objects in a virtual space: a comparative analysis between image and sound spatial representation and synthesis. In *Proceedings of the 5th Brazilian Symposium on Computer Music*, August 1998.

[Natkin00] S. Natkin. Mapping a virtual space into a real visual space. In *Proceedings of the 2000 International Computer Music Conference*, September 2000.

[Natkin01] S. Natkin, F. Schaeffer, and A. Topol. Functional specification of a distributed and mobile architecture for virtual sound space systems. In *Proceedings of the International Computer Music Conference (ICMC'2001)*, September 2001.

[Nielsen90] J. Nielsen. The art of navigating through hypertext. *Communications of the ACM*, 33(3), 1990.

[Nye93] A. Nye. *X Toolkit Intrinsics Programming Manual.* O'Reilly, 3rd edition, April 1993.

[Obeysekare96] U. Obeysekare, C. Williams, J. Durbin, L. Rosenblum, R. Rosenberg, F. Grinstein, R. Ramaturi, A. Landsberg, and W. Sandberg. Virtual workbench - a non-immersive virtual environment for visualizing and interacting with 3d objects for scientific visualization. In *Proceedings of the Visualization Conference*, San Francisco, CA, USA, 1996.

[Olsen93] K.A. Olsen, R.R. Korfhage, K.M. Sochats, M.B. Spring, and J.G. Williams. Visualisation of a document collection: The vibe system. *Information Processing and Management*, 29(1):69–81, 1993.

[Pearson93] Pearson and Steinmetz. The jove project: 3d tree-mapped universes and data sphere navigation, a method to increase the density of hierarchical information visualization in a finite display space, 1993.

[Pfister98] Hans-Rüdiger Pfister, Christian Schuckmann, Jennifer Beck-Wilson, and Martin Wessner. The metaphor of virtual rooms in the cooperative learning environment CLear. *Lecture Notes in Computer Science*, 1370:107–??, 1998.

[Picard97] R. Picard. *Affective Computing.* MIT Press, 1997.

[Plénacoste01] P. Plénacoste, E. Lecolinet, S. Pook, C. Dumas, and J.-D. Fekete. Zoomable and 3d representations for digital libraries. September 2001.

[Pook00] S. Pook, E. Lecolinet, G. Vaysseix, and E. Barillot. Des aides transparentes de navigation et un nouveau type de menu pour les interfaces zoomables. In *Proceedings of the AVI 2000 Conference*, pages 227–231, 2000.

[Price93] B.A. Price, R.M. Baecker, and I.S. Small. A principled taxonomy of software visualisation. *Journal of Visual Languages and Computing*, 4:211–266, 1993.

[Quinn79] N. Quinn and M. Breur. A force directed component placement procedure for printed circuit boards. *IEEE Transactions on Circuits and Systems*, 26(4):377–388, 1979.

[Rauber99a] A. Rauber and H. Bina. A metaphor graphics based representation of digital libraries on the www: Using the libviewer to make metadata visible. In *DEXA Workshop*, 1999.

[Rauber99b] A. Rauber and D. Merkl. Somlib: A digital library system based on neural networks. In *Proceedings of the ACM Digital Libraries 1999 Conference*, 1999.

[Reiss94] S.P. Reiss. 3-d visualization of program information. In *Proceedings of GD'94: DIMACS international workshop on graph drawing*, pages 12–24, Princeton, New Jersey, USA, October 1994.

[Rekimoto93] J. Rekimoto and M. Green. The information cube: Using transparencty in 3d information visualization. In *Proceedings of the Third Annual Workshop on Information Technologies & Systems (WITS'93)*, pages 125–132, 1993.

[Rezzonico96] S. Rezzonico and D. Thalmann. Browsing 3d bookmarks in bed. In *Proceedings of the WebNet96 Conference*, San Francisco, California, USA, October 1996.

[Richardson98] T. Richardson, Q. Staffont-Fraser, K.R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, January 1998.

[Risden00] K. Risden, M. P. Czerwinski, T. Munzner, and D. Cook. An initial examination of ease of use for 2D and 3D information visualizations of web content. *International Journal of Human Computer Studies*, 53(5):695–714, November 2000.

[Rivera96] K. Rivera, N. J. Cooke, and J. A. Bauhs. The effects of emotional icons on remote communication. In *Proceedings of ACM SIGCHI'96 Conference on Human Factors in Computing Systems*, Vancouver, Canada, April 1996. ACM Press.

[Rivlin94] E. Rivlin, R. Botafogo, and B. Scneiderman. Navigating in hyperspace: Designing a structured-based toolbox. *Communications of the ACM*, 37(2):87–96, 1994.

[ROBERT98] L. Robert and E. Lecolinet. Browsing hyperdocuments with multiple focus+context views. In *Proceedings of the UK Conference oh Hypertext*, 1998.

[ROBERT01] Laurent Robert. *Annotation et visualisation interactives de documents hypermédias*. Ecole Nationale Supérieure des Télécommunications, June 2001.

[ROBERTSON91] G.G. Robertson, J.D. Mackinlay, and S.K. Card. Cone trees: animated 3d visualizations of hierarchical information. *Communications of the ACM*, 36(4):57–71, April 1991.

[ROBERTSON93A] G.G. Robertson, S.K. Card, and J.D. Mackinlay. Information visualization using 3d interactive animation. *Communication of the ACM*, 36(4):57–71, April 1993.

[ROBERTSON93B] G.G. Robertson and J.D. Mackinlay. The document lens. In *Proceedings of the ACM Symposium on User Interface Software and Technology*. ACM Press, 1993.

[ROBERTSON98] G. Robertson, M. Czerwinski, K. Larson, D. Robbins, D. Thiel, and M. van Dantzich. Data mountain: Using spatial memory for document management. In *Proceedings of the 1998 ACM Conference on User Interface Software and Technology*, pages 153–162. ACM Press, November 1998.

[ROBERTSON00] G. Robertson, M. van Dantzich, D. Robbins, M. Czerwinski, K. Hinckley, K. Risden, D. Thiel, and V. Gorokhovsky. The task gallery: A 3d window manager. In *Proceedings of SIGCHI'2000 Conference on Human Factors in Computing Systems*, pages 494–501, The Hague, Netherlands, April 2000. ACM Press.

[ROMAN93] G. Roman and K.C. Cox. A taxonomy of program visualisation systems. *IEEE Computer*, December 1993.

[ROUSSEL01] N. Roussel. Exploring new uses of video with videospace. In *Proceedings of EHCI'01, the 8th IFIP International Conference on Engineering for Human-Computer Interaction*, pages 73–90. Springer, September 2001.

[RUBIN96] R.S. Rubin. Hotsauce lets users fly through the web. *MacWEEK Gateways*, 10(36), September 1996.

[RUDDLE97] R.A. Ruddle, S.J. Payne, and D.M. Jones. Navigating buildings in desktop virtual environments: Experimental investigations using extended navigational experience. *Experimental Psychology*, 3:143–157, 1997.

[SARKAR92] M. Sarkar and M.H. Brown. Graphical fisheye views of graphs. In *Proceedings of SIGCHI'92 Conference on Human Factors in Computing Systems*, pages 83–91, Monterey, CA, USA, May 1992. ACM Press.

[SARKAR94] M. Sarkar and M. H. Brown. Graphical fisheye views. *Communications of the ACM*, 37(12):73–84, 1994.

[SAVIOJA97] L. Savioja, J. Huopaniemi, T. Lokki, and R. Väänänen. Virtual environment simulation - advances in the diva project. In *Proceedings of the ICAD'97 Conference*, Palo Alto, CA, USA, November 1997.

[SCHAFFER96A] D. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, S. Dubs, and M. Roseman. Comparing fisheye and full-zoom techniques for navigating large information spaces. *ACM Transactions on Information Systems*, 1996.

[SCHAFFER96B] Doug Schaffer, Zhengping Zuo, Saul Greenberg, Lyn Bartram, John Dill, Shelli Dubs, and Mark Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Transactions on Computer-Human Interaction*, 3(2):162–188, 1996.

[SCHEIRER99] E.D. Scheirer, R. Väänänen, and J. Huopaniemi. Audiobifs: The mpeg-4 standard for effects. In *Processing of DAFXŠ98 Conference*, Barcelona, Spain, November 1999.

[SEBRECHTS99] M.M. Sebrechts, J. Vasilakis, M.S. Miller, J.V. Cugini, and S.J. Laskowski. Visualization of search results: A comparative evaluation of text, 2d and 3d interfaces. In *Proceedings of the 22nd ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–10. ACM Press, 1999.

[SHAW92] C. Shaw, J. Liang, M. Green, and Y. Sun. The decoupled simulation model for virtual reality systems. In *Proceedings of SIGCHI'92 Conference on Human Factors in Computing Systems*, pages 321–328, Monterey, CA, USA, May 1992. ACM Press.

[SHEELAGH97] M. Sheelagh, T. Carpendale, D.J. Cowperthwaite, and F.D. Fracchia. Extending distortion viewing from 2d to 3d. *Computer Graphics and Applications*, pages 42–51, 1997.

[SHNEIDERMAN80] B. Shneiderman. *Software Psychology.* Cambridge MA: Winthrop Publisher Inc., 1980.

[SHNEIDERMAN91] B. Shneiderman. Tree visualization with tree-maps: A 2-d space-filling approach. Department of Computer Science and HCI Laboratory, University of Maryland, June 1991.

[SMITH98] J. Smith, T. White, C. Dodge, D. Allport, J. Paradiso, and N. Gershenfeld. Electric field sensing for graphical interfaces. *IEEE Computer Graphics and Applications*, 18(3):54–60, May 1998.

[SNOWDON96] D. Snowdon, L. Fahlèn, and M. Stenius. Www3d: A 3d multi-user web browser. In *WebNet96 Proceedings*, San Francisco, California, USA, October 1996.

[STANDISH84] T.A. Standish. An essai on software reuse. *IEEE transactions on Software Engineering*, 10(5):494–497, September 1984.

[STASKO92] J.T. Stasko and C. Patterson. Understanding and characterizing software visualisation systems. In *Proceedings of the IEEE 1992 workshop on Visual Languages*, pages 3–10, Seattle, Washington, USA, 1992.

[STOAKLEY95] R. Stoakley, M. J. Conway, and R. Pausch. Virtual reality on a wim: Interactive worlds in miniature. In *Proceedings of SIGCHI'95 Conference on Human Factors in Computing Systems*, Denver, CO, USA, May 1995. ACM Press.

[STOKOWSKI02] P. Stokowski. Interactions 3d dans un poste de lecture sous virtools dev. Mémoire de DEA, Conservatoire National des Arts et Métiers, 2002.

[STONE94] M. C. Stone, K. Fishkin, and E. A. Bier. The movable filter as a user interface tool. In *Proceedings of SIGCHI'94 Conference on Human Factors in Computing Systems*, pages 306–312, Boston, MA, USA, April 1994. ACM Press.

[STOREY95] M-A. Storey and H. Müller. Manipulating and documenting software structures using shrimp views. In *Proceedings of the ICSM'95 conference on Software Maintenance*, pages 275–284, Nice, France, October 1995.

[STOREY97] M.-A. D. Storey, K. Wong, and H.A. Muller. Rigi: A visualization environment for reverse engineering. In *International Conference on Software Engineering*, pages 606–607, 1997.

[TEITELMAN86] W. Teitelman. Ten years of window systems - a retrospective view. In *Methodology of window management*, pages 35–46. Springer-Verlag, 1986.

[TOPOL97] A. Topol, C. Thiria, and Y. Creach. Simulateur urbain. Rapport de valeur C Conception d'Applications Multimédias, Conservatoire National des Arts et Métiers, September 1997.

[TOPOL01] A. Topol. Vrml : étude, mise en oeuvre et applications. Mémoire d'ingénieur, Conservatoire National des Arts et Métiers, July 2001.

[TSINGOS97] N. Tsingos and J.-D. Gascuel. Sounftracks for computer animation: Sound rendering in danymic environments with occlusions. In *Graphics Interface '97*, May 1997.

[TSOU98] H. J. Fang Tsou and C. Chang. A multiuser 3d web browsing system. *IEEE Internet Computing*, 2(5):70–79, 1998.

[UTTING89] . Utting and N. Yankelovitch. Context and orientation in hypermedia network. *ACM Transaction on Information System*, 7(1):58–84, 1989.

[VRML97] Virtual reality modeling language - international standard. Technical report, VRML Consortium, 1997. ISO/IEC 14772-1:1997.

[Walker93] G. Walker, P.A. Rea, S. Whalley, M. Hinds, and N.J. Kings. Visualisation of telecommunications network data. *British Telecommunications Technology Journal*, 11(4):54–63, October 1993.

[Walker95] G. Walker. Challenges in information visualisation. *British Telecommunications Engineering Journal*, 14:17–25, April 1995.

[Waller98] D. Waller, E. Hunt, and D. Knapp. The tranfer of spatial knowledge in virtual environment training. *Presence*, 7:129–144, 1998.

[Ware93] C. Ware, D. Hui, and G. Franck. Visualizing object oriented software in three dimension. In *Proceedings of CASCON'93*, pages 612–620, Toronto, Ontario, Canada, October 1993.

[Ware94] C. Ware and G. Franck. Viewing a graph in a virtual reality display is three times as good as a 2d diagram. In *Proceedings of the 1994 IEEE Conference on Visual Languages*, pages 182–183, St. Louis, Missouri, USA, October 1994.

[Warusfel95] O. Warusfel and F. Cruz-Barney. Validation of a computer simulation environment for room acoustics prediction. In *Proceedings of the 15th International Conference on Acoustics*, Trondheim, 1995.

[Watlington95] J. A. Watlington, M. Lucente, C. J. Sparrell, V. M. Bove, and I.Tamitani. A hardware architecture for rapid generation of electroholographic fringe patterns. In *proceedings of SPIE'95*, 1995.

[Watt93] A. Watt. *3D Computer Graphics*. Addison-Wesley, New York, 1993.

[Wood95] A. M. Wood, N. S. Drew, R. Beale, and R. J. Hendley. Hyperspace: Web browsing with visualisation. In *Third International World-Wide Web Conference Poster Proceedings*, pages 21–25, Darmstadt, Germany, April 1995.

[Young96] P. Young. Three dimensional information visualisation. Technical report, Internal Report, Centre for Software Maintenance, University of Durham, March 1996.

[Zeleznik91] R. C. Zeleznik, D. Conner, M. Wloka, D. Aliaga, N. Huang, P. Hubbard, B. Knep, H. Kaufman, J. F. Hughes, and A. Van Dam. An object-oriented framework for the integration of interactive animation techniques. *Computer Graphics*, 25(4), July 1991.

[Zeleznik97] R. C. Zeleznik, D. Brookshire Conner, A. S. Forsberg, and P. S. Strauss. Two pointer input for 3d interaction. In *Proceedings of 1997 Symposium on Interactive 3D Graphics*, Providence, Rhode Island, USA, April 1997.

[Zeleznik99] R. C. Zeleznik and A. S. Forsberg. Unicam - 2d gestural camera controls for 3d environmentsn. In *Proceedings of 1999 Symposium on Interactive 3D Graphics*, Atlanta, GA, USA, 1999.

[ZHAI94] S. Zhai, W. Buxton, and P. Milgram. The silk cursor: Investigating transparency for 3d target acquisition. In *Proceedings of SIGCHI'94 Conference on Human Factors in Computing Systems*, Boston, MA, USA, April 1994. ACM Press.

# *Liste des acronymes*

| | |
|---|---|
| ABU | Association des Bibliophiles Universels |
| AFAP | *As Fast As Possible* |
| ALUT | *OpenAL Utility Toolkit* |
| API | *Application Programming Interface* |
| ASCII | *American Standard Code Information Interchange* |
| AVI | *Audio Video Interleaved* |
| AVO | *Audio Visual Object* |
| AWT | *Abstract Window Toolkit* |
| BIFS | *Binary Format For Scenes* |
| BSP | *Binary Space Partition* |
| CALS | *Continuous Acquisition and Life-Cycle Support* |
| CAO | Conception Assistée par Ordinateur |
| CEDRIC | Centre d'Études et de Recherche en Informatique du CNAM |
| CGI | *Common Gateway Interface* |
| CNAM | Conservatoire National des Arts et Métiers |
| CNUM | Conservatoire NUMérique des arts et métiers |
| DAI | *DMIF Application Interface* |
| DEM | *Digital Elevation Model* |
| DMIF | *Delivery Multimedia Integration Framework* |
| DNI | *DMIF Network Interface* |
| DRI | *Direct Rendering Infrastructure* |
| DTD | *Document Type Definition* |
| EAX | *Environmental Audio Graphical Librarian Editor* |
| EAX | *Environmental Audio eXtensions* |
| FDP | *Force Directed Placement* |
| FLL | *Fahrenheit Low Level* |
| FLM | *Fahrenheit Large Model* |
| FPS | *Frames Per Second* |
| FSG | *Fahrenheit Scene Graph* |
| GLU | *openGL Utility* |
| GLUT | *openGL Utility Toolkit* |
| GLX | *openGL eXtension* |
| GNU | *GNU is Not Unix* |
| GPL | *GNU General Public Licence* |
| GPU | *Graphic Processor Unit* |

| | |
|---|---|
| GUI | *Graphical User Interface* |
| HAL | *Hardware Abstraction Layer* |
| HEL | *Hardware Emulation Layer* |
| HTML | *HyperText Markup Language* |
| HTTP | *HyperText Transport Protocol* |
| ICD | *Installable Client Driver* |
| IEC | *International Electrotechnical Commission* |
| IRCAM | Institut de Recherche et Coordination Acoustique/Musique |
| ISO | *International Organization for Standardization* |
| IHM | Interaction Homme-Machine |
| JNI | *Java Native Interface* |
| LOD | *Level Of Details* |
| MARC | *MAchine Readable Cataloguing* |
| MCD | *Mini Client Driver* |
| MMX | *Matrix Math eXtension* |
| MPEG | *Motion Picture Expert Group* |
| MTS | *MeTaStream* |
| OpenAL | *Open Audio Library* |
| OpenGL | *Open Graphic Library* |
| PAO | Publication Assistée par Ordinateur |
| PDF | *Portable Document Format* |
| PIT | *Populated Information Terrain* |
| POO | Programmation Orientée Objet |
| POV | *Persistence Of Vision* |
| RAMDAC | *Random Access Memory Digital-Analog Converter* |
| RGB | *Red Green Blue* |
| SDK | *Software Development Kit* |
| SGI | *Silicon Graphics Inc.* |
| SGML | *Standard Generalized Markup Language* |
| SMTP | *Simple Mail Transfer Protocol* |
| SSE | *Streaming SIMD Extensions* |
| TEI | *Text Encoding Initiative* |
| TCAO | Travail Coopératif Assisté par Ordinateur |
| TCP/IP | *Transmission Control Protocol/Internet Protocol* |
| URL | *Uniform Resource Locator* |
| VRML | *Virtual Reality Modelling Language* |
| WIM | *Windows In Miniature* |
| WIMP | *Windows Icons Menus Pointer* |
| WWW | *World Wide Web* |
| X3D | *eXtensible 3D* |
| XML | *eXtensible Murkup Language* |

# *Liste des figures*


2.1 Un exemple de visualisation avec AMAZE [Boyle96]. . . . . 18
2.2 Un exemple de visualisation circulaire avec Winona. . . . . 19
2.3 Un exemple de visualisation avec Q-PIT [Benford94b]. . . . . 21
2.4 Un exemple de visualisation avec BEAD [Chalmers92]. . . . . 22
2.5 Un exemple de visualisation en 2D avec VIBE [Olsen93]. . . . . 23
2.6 Un exemple de visualisation en 2D avec VR-VIBE. . . . . 24
2.7 Un exemple de visualisation en 3D avec VR-VIBE. . . . . 24
2.8 Les cônes de navigation de LyberWorld [Hemmje94]. . . . . 25
2.9 La sphère de pertinence de LyberWorld [Hemmje94]. . . . . 26
2.10 La métaphore de galaxie de Vineta [Krohn96b]. . . . . 28
2.11 La métaphoren du paysage de Vineta [Krohn96b]. . . . . 28
2.12 Les visualisations 2D de VR-Mapper [Benford93]. . . . . 30
2.13 La visualisation 3D de type *cone tree* de VR-Mapper [Benford93]. . . . . 30
2.14 Exemple d'hyperstructure. . . . . 31
2.15 Hyperspace : Narcissus utilisé pour visualiser le Web [Wood95]. . . . . 32
2.16 Une collection de pages HTML avec leurs liens avec WWW3D [Snowdon96]. 33
2.17 La représentation d'une page HTML avec WWW3D [Snowdon96]. . . . . 34
2.18 La visualisation SemNet [Fairchild88]. . . . . 35
2.19 Le *Web Forager* [Card96]. . . . . 36
2.20 Un *Web Book* [Card96]. . . . . 37
2.21 La *Data Mountain* ([Robertson98]). . . . . 38
2.22 Visualisation d'une structure de fichiers par FSN. . . . . 42
2.23 La gestion des versions et l'exploration de classes avec VOGUE. . . . . 43
2.24 La visualisation de programmes parallèles avec VisualLinda. . . . . 44
2.25 Plusieurs vues d'un réseau en utilisant GV3D [Ware93]. . . . . 45
2.26 Représentations de structures avec Narcissus [Hendley95b]. . . . . 46
2.27 Une vue zoomable d'une structure logicielle avec SHriMP [Storey95]. . . . 47
2.28 Le MEMEX de Vannevar Bush. . . . . 51
2.29 Les dispositifs d'entrée imaginés par V. Bush. . . . . 51
2.30 L'interface de libViewer [Rauber99a]. . . . . 55
2.31 La métaphore du tunnel de MaW3D [Leach97]. . . . . 56
2.32 Le gestionnaire de fenêtres 3DWM. . . . . 58
2.33 La métaphore du tunnel de la TaskGallery [Robertson00]. . . . . 59
2.34 Les options de navigation dans la TaskGallery [Robertson00]. . . . . 59
2.35 La palette d'applications de la TaskGallery [Robertson00]. . . . . 60
2.36 Le positionnement des bureaux dans la TaskGallery [Robertson00]. . . . . 60

3.1 Les paradigmes d'interface utilisateur. . . . . 62
3.2 Une vision de poisson ou *Fish-eye View* [Furnas86]. . . . . 68
3.3 Un exemple d'arbre hyperbolique [Lamping95]. . . . . 70
3.4 La loupe à documents de Xerox [Robertson93b]. . . . . 71
3.5 Une vue fractale [Koike93]. . . . . 71
3.6 Le navigateur HTML de PAD++ [Bederson96]. . . . . 73
3.7 Un exemple de vue filtrée [Sarkar92]. . . . . 73
3.8 Un exemple de filtre local [Stone94]. . . . . 74
3.9 Exemple de loupes magiques [Bier93]. . . . . 75
3.10 Un exemple de graphique 3D. . . . . 78
3.11 Un espace de Benedikt [Benedikt91]. . . . . 79
3.12 Le *Perspective Wall* de Xerox [Mackinlay91]. . . . . 81
3.13 Le *Cam Tree* de Xerox [Robertson91]. . . . . 82
3.14 Visualisation sphérique produite par VizNet [Fairchild93]. . . . . 83
3.15 Une visualisation de type *Information Cube* [Rekimoto93]. . . . . 84
3.16 Exemple de *widgets* de navigation. . . . . 86
3.17 Les différentes couches d'une architecture 3D. . . . . 87
3.18 Fonctions accélérées par les trois générations de cartes graphiques 3D. . . . 90
3.19 Processus de création et d'exécution d'une applications 3D. . . . . 92
3.20 Chaînes de production et de visualisation d'une scène 3D. . . . . 94
3.21 Architecture et pipeline d'OpenGL. . . . . 96
3.22 Position des différentes librairies OpenGL. . . . . 97
3.23 Architecture logicielle et pipeline graphique de Direct3D. . . . . 100
3.24 Exemple simple de graphe Java3D. . . . . 102
3.25 L'architecture de Fahrenheit. . . . . 104
3.26 Les composants d'un co-navigateur 3D. . . . . 105
3.27 Les différentes couches de l'architecture MPEG4. . . . . 109
3.28 Une copie d'écran du logiciel Virtools Dev. . . . . 113
3.29 Un script comportemental avec Virtools Dev. . . . . 114
3.30 Associations des événement avec un code à exécuter. . . . . 116
3.31 Exemple de graphe de scène VRML avec appel de script. . . . . 117
3.32 Un échogramme schématique prenant en compte l'effet de salle. . . . . 125
3.33 Les deux méthodes de spatialisation sonore. . . . . 126
3.34 La synthèse sonore avec le modèle physique [Funkhouser98]. . . . . 127
3.35 Les paramêtres de spatialisation du Spat de l'Ircam [Jot95]. . . . . 128

4.1 Les modules du simuateur urbain. . . . . 132
4.2 L'architecture DRI de XFree86 4.0. . . . . 134
4.3 Le fonctionnement de notre interpréteur Yacc/Lex. . . . . 137
4.4 La gestion des capteurs dans notre navigateur VRML. . . . . 138
4.5 La gestion de la mémoire allouée pour les textures avec Java. . . . . 138
4.6 Les WIMs [Stoakley95] de notre navigateur VRML. . . . . 140
4.7 Exemple d'aides à la navigation : les zooms. . . . . 141
4.8 Une copie d'écran de la biliothèque numérique ABU. . . . . 144
4.9 La recherche d'occurences envisageable avec l'ABU en mode graphique. . . . 145
4.10 Une partie de la donation de Sartiaux. . . . . 148

4.11 Une interface zoomable en HTML. . . . . 149
4.12 Classement et parcours possibles pour une recherche sur 3 critères. . . . . 152
4.13 La roue à livres de Ramelli. . . . . 154
4.14 Interfaces en “atomium” et en “cylindre”. . . . . 155
4.15 Placement des groupes de livres dans une sphère. . . . . 155
4.16 Les interfaces de consultation du CNUM en 2D et en 3D. . . . . 159
4.17 Architecture du poste de lecture. . . . . 160
4.18 Notre prototype de médiathèque. . . . . 161
4.19 La première ébauche du deuxième prototype de poste de lecture. . . . . 164
4.20 La maquette sous Virtools du poste de lecture. . . . . 165
4.21 Le livre et ses marques-pages. . . . . 165
4.22 Les fonctionnalités du trépied. . . . . 166
4.23 La métaphore du livre 3D avec son trépied. . . . . 167
4.24 Une représentation du panier au premier plan. . . . . 168
4.25 La métaphore du “cockpit”. . . . . 170
4.26 La métaphore du “camembert”. . . . . 171
4.27 La métaphore “sans dessus-dessous”. . . . . 173
4.28 Applications 2D dans notre bibliothèque 3D. . . . . 174
4.29 Une application 3D dans un gestionnaire de fenêtres 2D. . . . . 176
4.30 Les applications 2D dans un environnement 3D. . . . . 177
4.31 Initialisation et gestion des interactions avec le contenu d’une fenêtre 3D. . 179
4.32 Exemple de scène virtuelle sonore. . . . . 181
4.33 Les graphes de scène comprenant des nœuds sonores avec MPEG4 et VRML. 183

5.1 Création d’une fenêtre dans le poste de lecture en VRML. . . . . 193
5.2 Suppression d’une fenêtre dans le poste de lecture en VRML. . . . . 194
5.3 Immobilité d’une fenêtre punaisée dans le poste de lecture en VRML. . . . . 195
5.4 Rangement d’une fenêtre dans le poste de lecture en VRML. . . . . 196
5.5 Changement de page dans le poste de lecture en VRML. . . . . 197
5.6 Exemple de scène sonore avec VRML. . . . . 198
5.7 Graphe de scène VRML avec comportements. . . . . 201
5.8 Taux de rafraichissement des moteurs de rendu et d’animations. . . . . 201
5.9 Exemple de composants différant par leur apparence. . . . . 203
5.10 Exemple de composants différant par leur apparence. . . . . 203
5.11 Mise en rapport des termes “comportement”, “interaction” et “animation”. . 204
5.12 Ordonnancement des comportements dans un graphe. . . . . 206
5.13 Politique de placement et comportements programmés. . . . . 208
5.14 Automate de comportements. . . . . 212
5.15 Exemple d’absorption sonore des géométries. . . . . 218
5.16 La chaîne de production binaurale des sons. . . . . 219

6.1 L’apport des rotations dans une interface 3D. . . . . 223
6.2 La couche d’abstraction matérielle DHAL. . . . . 228
6.3 L’architecture pressentie de la deuxième version du bureau 3D. . . . . 230

# *Liste des tableaux*

2.1 Les tâches de haut niveau décrites par L. Robert [Robert01]. . . . . 9
2.2 Les outils d'aide à l'exploration. . . . . 11
2.3 Les services pour accéder aux données d'un système de visualisation. . . . . 12
2.4 Les trois types d'interfaces. . . . . 12
2.5 Les principales expressions rencontrées dans le domaine de la visualisation de logiciels. . . . . 41
2.6 Les quatre activités d'une bibliothèque. . . . . 50

5.1 Les politiques de placement (*layouts*) de l'AWT de Java. . . . . 208

Thèse réalisée avec

LaTeX

sur système

Gnu/Linux