



Interaction et contexte dans les interfaces zoomables

Stuart Pook

► To cite this version:

Stuart Pook. Interaction et contexte dans les interfaces zoomables. Interface homme-machine [cs.HC].
Télécom ParisTech, 2001. Français. NNT : . tel-00005779

HAL Id: tel-00005779

<https://pastel.hal.science/tel-00005779>

Submitted on 5 Apr 2004

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Interaction and Context in Zoomable User Interfaces

Interaction et contexte dans les interfaces zoomables

by

Stuart Pook

Thesis Presented for the Degree of Doctor of the
École Nationale Supérieure des Télécommunications
(Network and Computer Science Department)
Paris, France

Defended 15 June 2001 before the jury consisting of

Joëlle Coutaz	Chair
Michel Beaudouin-Lafon	Readers
Guy Melançon	
Éric Lecolinet	Thesis Director
Emmanuel Barillot	Examiners
Jean-Daniel Fekete	
Claude Kintzig	

This thesis was written as part of the Zomit project. More information on the Zomit project can be found at the URL <http://www.infobiogen.fr/services/zomit/> or at the URL <http://www.infres.enst.fr/net/zomit/>. These pages have links to a downloadable version of this thesis, to copies of all the papers published as part of this project, and to online demonstrations of the software developed during the thesis.

Stuart Pook can be contacted by email at the address stuart@acm.org or via the web page <http://www.acm.org/~stuart/>. The supervisor of this thesis, Éric Lecolinet, can be contacted by email at the address elc@enst.fr and his home page is <http://www.enst.fr/~elc/>.

This document was prepared with L^AT_EX, netpbm, xfig, ghostscript, xpdf and many other free software tools.

This thesis was published by the École Nationale Supérieure des Télécommunications (ENST or Télécom Paris), 46 rue Barrault, 75634 Paris cedex 13, France (<http://www.enst.fr>). This copy of the thesis was formatted on 14 March 2002.

Copyright © 2002 ENST. Thesis number: ENST 2001 E 024

Acknowledgements

This doctoral thesis was the result of a collaboration between Infobiogen, France Télécom R&D, and the École Nationale Supérieure des Télécommunications.

None of the work presented here would have been possible without the help and ideas of my thesis director Éric Lecolinet. For his unfailing support, even at midnight when presented with my attempts at writing French, my sincere thanks.

My colleagues at Infobiogen, especially Emmanuel Barillot and Guy Vaysseix, helped me define my area of research, aided me in finding the funding for the project, and supported my work throughout my thesis. I would also like to thank Laurence Samarcq for proofreading many of the articles that I wrote in French.

Gérard Poulain and Claude Kintzig from France Télécom helped organise the funding of my thesis and showed their continuing interest in my work.

I would like to express my appreciation to the members of the jury: Emmanuel Barillot, Michel Beaudouin-Lafon, Joëlle Coutaz, Jean-Daniel Fekete, Claude Kintzig, and Guy Melançon. I thank Guy Melançon and Michel Beaudouin-Lafon for their efforts as the examiners of my thesis and Joëlle Coutaz for agreeing to be president of the jury.

Jean-Daniel Fekete was the first person to use Zomit to develop a Zoomable User Interface. I am grateful for his patience and his comments that helped me improve Zomit.

I gratefully acknowledge Isabelle Demeure's help in understanding the administration of Télécom Paris. The members of the secretariat, Chantal Jimenez, Jean-Louis Poirot, and Hayette Soussou, made my dealings with that administration so much easier. Philippe Dax, Serge Gadret and Patrick Clément looked after and helped me use the department's computers.

I must also thank my fellow doctoral student at Télécom Paris, Laurent Robert for sharing his office and the stress of doing a thesis.

This work was financed by the European Union (contract BIO4-CT96-0346) and by France Télécom R&D (contract 97 754 21).

Abstract

Many interactive computer systems use menus as an important part of their interface. Menus allow users to select operations but not to control their execution. A second interactor, such as a dialog box, has to be used to control the chosen operation and thus complete the interaction. This decomposition of what is a single action from the user's point of view into two distinct steps slows down interaction with computer systems. This thesis proposes a new contextual pop-up menu, called a *Control Menu*, that includes proportional control of the chosen operation with immediate feedback. Using this menu gives a more fluid control of complex interfaces and has the advantage of an expert form of use that is very similar to, and thus easily learnt from, the novice usage.

Interaction with databases and navigation within large information spaces are important tasks in many applications. Many visualization systems cause user disorientation as users find it difficult to understand their position within the information space and to locate desired information. This thesis proposes several new contextual aids for Zoomable User Interfaces that address these issues.

The first aid, a hierarchical view of the information space, helps users understand their current position and the location of the desired information, and accelerates navigation. The second type of aid uses dynamically generated transparent and temporary views that are created and controlled by users in a single gesture. These interactive views overlay the current view of the focus with contextual or historical information which shows users what surrounds the current view or the route taken to arrive at that view. The effective use of these new aids requires a tight coupling between interaction and presentation which is achieved via the use of *Control Menus*.

Résumé

La plupart des logiciels interactifs font un usage intensif des systèmes de menus. Ceux-ci permettent de sélectionner des opérations variées mais n'offrent généralement pas de moyen de contrôler leur exécution. Un second interacteur doit alors être utilisé (typiquement une boîte de dialogue) ce qui impose une décomposition d'une action unique (du point de vue de l'utilisateur) en une suite d'interactions successives. Nous proposons un nouveau type de menu, appelé *Control Menu*, qui permet de fluidifier l'interaction en unifiant la sélection et le contrôle des opérations.

L'accès interactif aux bases de données et la navigation dans les espaces d'information de grande taille constituent des tâches primordiales pour de nombreuses applications. Or, les systèmes de visualisation posent souvent des problèmes de désorientation, les utilisateurs ayant fréquemment des difficultés à se localiser précisément dans l'espace d'information et à trouver les données recherchées. Cette thèse propose plusieurs techniques d'aide contextuelle pour remédier à ces problèmes dans le cadre des interfaces basées sur le concept de zoom sémantique (ou *interfaces zoomables*).

Le premier type d'aide, qui offre une vue « en profondeur » de l'espace d'information via une représentation hiérarchique, permet non seulement de faciliter la localisation de la position courante et des informations recherchées mais aussi d'accélérer la navigation. La seconde technique est basée sur la génération dynamique de vues transparentes et temporaires que les utilisateurs peuvent créer et contrôler en un seul geste. Ces vues interactives se superposent à la vue courante en y rajoutant des informations contextuelles ou historiques qui aident l'utilisateur à comprendre ce qui entoure le point de focus ou quel chemin a été effectué pour arriver à ce dernier. Ces aides contextuelles nécessitent un couplage étroit entre interaction et présentation, qui est obtenu en utilisant des *Control Menus*.

Contents

1	Introduction	21
1.1	Improving Interaction	21
1.2	Adding Context	22
I	Interactors	25
2	Today's Menus and Controlling Operations	27
2.1	Existing Menus	27
2.1.1	Pull-Down and Drop-Down Menus	28
2.1.2	Pop-up Menus	29
2.1.3	Pie Menus	29
2.1.4	Marking Menus	30
2.1.5	Command Compass	31
2.1.6	FlowMenus	32
2.1.7	Fisheye Menus	34
2.2	Taxonomy	36
2.2.1	Menu Placement	36
2.2.2	Visual Representation	38
2.2.3	Size and Shape of Menu Items	38
2.2.4	Menu Shape	40
2.2.5	Visible Versus Invisible Menus	41
2.2.6	Interactor Distance	42
2.2.7	Contextual	42
2.2.8	Keyboard Accelerators	44
2.2.9	Self-Revealing	44
2.2.10	Novice and Expert Gestures Similar	45
2.2.11	Combines Selection and Control	46
2.2.12	Summary	46
2.3	Controlling Operations	46

2.4	Problems With Existing Interactors	48
2.5	Example: the Scale in Acrobat Reader	48
3	A New Interactor	53
3.1	Control Menu	54
3.1.1	Pop-up Menu	54
3.1.2	Contextual Menu	54
3.1.3	Circular Menu	55
3.1.4	Menu Visible Only for Novices	55
3.1.5	Novice and Expert Modes Similar	56
3.1.6	Menu is Transparent	56
3.1.7	Activation Based on Distance	56
3.1.8	Discrimination Based on Angle	57
3.1.9	Control of the Operation	58
3.1.10	Comments	59
3.2	Controlling Operations	59
3.2.1	Operations With One Parameter	59
3.2.2	Operations With Two Parameters	61
3.2.3	Comments	63
3.3	Sub-menus	65
3.4	Buttons	66
3.5	More Than a Menu	67
3.5.1	Interaction Model	67
3.5.2	Fitts' Law	68
3.6	Applications	69
3.6.1	Zoomable User Interfaces	69
3.6.2	Interaction in a Virtual World	69
3.6.3	Text and Presentation Editors	70
3.6.4	Interfaces With Limited Screen Space	72
3.7	Discussion	72
II	Visualization	73
4	Visualization Research	75
4.1	Distorted Views	76
4.1.1	Fisheye Views	77
4.1.2	Perspective Wall	77
4.1.3	Document Lens	80
4.1.4	Rubber Sheets	81
4.1.5	3D Pliable Surfaces	83

4.1.6	Table Lens	84
4.1.7	Hyperbolic Display	86
4.1.8	Linear and Non-Linear Transformations	88
4.2	Visualizing Hierarchies	89
4.2.1	gIBIS	90
4.2.2	Clustered Hierarchies	92
4.2.3	ZoomTree	94
4.2.4	Treemaps	95
4.2.5	Cone and Cam Trees	96
4.2.6	Multitrees	97
4.2.7	Discussion	98
4.3	Transparent Tools	100
4.3.1	Introduction	100
4.3.2	Semi-Transparent Tool Palettes	102
4.3.3	Semi-Transparent 3D Cursors	103
4.3.4	Translucent Patches	103
4.3.5	Toolglass Widgets	105
4.3.6	Magic Lenses	106
4.3.7	Transparent Overview Layers	111
4.3.8	Excentric Labelling	113
4.3.9	Discussion	114
4.4	Zoomable User Interfaces	115
4.4.1	Semantic Zooming	115
4.4.2	Special Objects in Zoomable User Interfaces	116
4.4.3	Pad++	118
4.4.4	NaviQue	119
4.4.5	Information Density	120
4.4.6	Macroscopic	122
4.4.7	Goal-Directed Zoom	123
4.4.8	Desert Fog	123
4.4.9	Discussion	125
4.5	Taxonomy	126
4.5.1	Taxonomy of Distortion-Oriented Techniques	126
4.5.2	“Task by Data Type” Taxonomy	128
4.5.3	Dynamics	130
4.5.4	Interaction	132
4.5.5	Design Patterns	134
4.5.6	Presentation Taxonomy	137
4.5.7	Deformation Taxonomy	138
4.6	Conclusion	141

5	New Context Aids for Zoomable User Interfaces	143
5.1	Hierarchy Trees	144
5.1.1	Zoomable User Interfaces Visualize Hierarchical Information Spaces	144
5.1.2	Make the Hierarchy Visible	144
5.1.3	Make the User's Position in the Hierarchy Visible	145
5.1.4	Using the Hierarchy Tree to Navigate	146
5.1.5	Comparison With Other Techniques	147
5.1.6	Limitations in Open Information Spaces	148
5.2	Context Layer	149
5.2.1	Context and Focus Both Visible	149
5.2.2	Context Layer is Temporary	149
5.2.3	Transparency Used	152
5.2.4	Context is Shown at Many Different Scales	152
5.2.5	No Optical or Positional Deformation	154
5.2.6	Distinguishing the Focus and the Context	154
5.2.7	Context Layer is Quasimodal	154
5.2.8	Synergy Between Interaction and Control	155
5.2.9	Example	157
5.2.10	Similar Techniques	157
5.3	History Layer	157
5.4	Conclusion	160
6	Zomit: a Development Tool for Zoomable User Interfaces	161
6.1	What is Zomit?	161
6.1.1	Navigation and Interaction	162
6.1.2	Client/Server Architecture for the Internet	164
6.1.3	Generic Tool	164
6.1.4	Creating the Virtual World	165
6.1.5	Lazy Evaluation	167
6.1.6	Positioning Regions and Objects	167
6.1.7	Portals and Lenses	168
6.1.8	Communication Server Side	169
6.1.9	Communication Client Side	170
6.1.10	Exchanges Between the Server and Client	170
6.2	Architecture Analysis	172
6.2.1	Architecture Modelling Techniques	172
6.2.2	Zomit Modelled by PAC-Amodeus	175
6.3	Implementation	178
6.3.1	Server	178
6.3.2	Generic Client in Java	179

6.3.3	Graphics in Java	180
6.4	Other Zoomable User Interfaces	184
6.4.1	Pad	184
6.4.2	Tabula Rasa	184
6.4.3	Pad++'s Architecture	184
6.4.4	Jazz's Architecture	186
6.5	Results and Perspectives	187
7	Zomit Applications	189
7.1	An Interface to a Genetic Database	189
7.1.1	Why a Zoomable User Interface?	189
7.1.2	HuGeMap Database	190
7.1.3	Using Zomit	191
7.1.4	ZoomMap: Zooming and Portals	192
7.1.5	ZoomMap's Magic Lenses	192
7.1.6	Zooming On Indices	196
7.1.7	Evaluation	196
7.2	CDI: a Zoomable Virtual Library	198
8	Conclusion and Perspectives	201
9	Synthèse	205
9.1	Introduction	205
9.1.1	Amélioration de l'interaction	205
9.1.2	Davantage de contexte	206
9.2	Interaction	207
9.2.1	Menus classiques et contrôle des opérations	208
9.2.2	Un nouvel interacteur	212
9.2.3	FlowMenus	220
9.3	Visualisation	222
9.3.1	Recherche en visualisation	222
9.3.2	Nouvelles aides de contexte pour les interfaces zoomables	232
9.3.3	L'outil de développement de ZUIs : Zomit	239
9.3.4	Applications de Zomit	242
9.3.5	Conclusion	242
9.4	Conclusion et perspectives	244
10	Publications	247
11	Bibliography	249

List of Tables

2.1	Menu summary	47
4.1	Simple tree visualization taxonomy	98
4.2	Classification following the taxonomy in Figure 4.51	127
4.3	Distortion methods taxonomy	141
4.4	Visualization taxonomy summary	141
9.1	Taxonomie de modes de déformation	231
9.2	Sommaire de la taxonomie de visualisation	231

List of Figures

2.1	Pull-down menu in Netscape	28
2.2	Pie menu and its active regions	30
2.3	Marking menu use	31
2.4	Hotbox	32
2.5	Fixed zoom factors in a FlowMenu	33
2.6	Variable zoom factor in a FlowMenu	34
2.7	100 item fisheye menu	35
2.8	Xerox Star function keys	37
2.9	Tear-off menus in Sun's mailtool	37
2.10	Xfig buttons and corresponding text	39
2.11	Pie menu in SimCity	39
2.12	Menu item selection discrimination	40
2.13	Gestures in Tivoli	41
2.14	Circular menu hierarchy and the associated marks	42
2.15	Contextual menu in Netscape Communicator	43
2.16	Buttons in Netscape Communicator	45
2.17	Degree of indirection of an interactor	47
2.18	Changing the scale in Acrobat Reader 3	49
2.19	Selecting the dezoom mode in Acrobat Reader 4	50
3.1	Control Menu in a Zoomable User Interface	53
3.2	Control Menu and two types of object	55
3.3	Activation based on distance	57
3.4	Discrimination based on angle	58
3.5	Zooming with a Control Menu	60
3.6	Panning with a Control Menu	62
3.7	Lens sub-menu in our Zoomable User Interface	66
3.8	Degree of indirection	67
3.9	Control Menu in vreng	70
3.10	Selecting text with a Control Menu	71

4.1	Fisheye view of a C program	78
4.2	Perspective Wall	79
4.3	Document Lens	80
4.4	Rubber Sheet with orthogonal stretching	81
4.5	Rubber Sheet with polygonal stretching	82
4.6	Single focus 3D pliable surface with flattened top	83
4.7	Magnification and distortion with 3D pliable surfaces	83
4.8	Double focus 3D pliable surface	84
4.9	Distortion patterns	84
4.10	Rarer distortion patterns	85
4.11	Table Lens	86
4.12	Hyperbolic display	87
4.13	Linear magnification	88
4.14	Linear and non-linear magnification	89
4.15	Directory browser	90
4.16	gIBIS interface	91
4.17	gIBIS node index window	91
4.18	Clustered hierarchies with 2 open clusters	92
4.19	Two nodes expanded to circuit diagram level	93
4.20	Hierarchically clustering a network	93
4.21	ZoomTree showing local focus+context views	94
4.22	Treemap	95
4.23	Cone Tree	96
4.24	Cam Tree	97
4.25	Multitree showing an information tree used by two professors	99
4.26	Centrifugal view centred on the node “directions”	100
4.27	Semi-transparent text menu	102
4.28	Semi-transparent tool palette	103
4.29	Semi-transparent 3D cursor	104
4.30	Translucent Patch containing a list drawn over a sketch	104
4.31	Calculator applied to a Translucent Patch	105
4.32	Colour changing Toolglass	106
4.33	Creating and positioning objects with a Toolglass	106
4.34	Overlapping Magic Lenses	107
4.35	Overlapping Magic Lenses on a city map	108
4.36	Magnifying Magic Lens	109
4.37	Magic Lens in two applications	109
4.38	Flat lens in 3D space	110
4.39	Volumetric lens in 3D space	111
4.40	Annotated 70% transparent overview layer in a pipeline system	112
4.41	Representations of objects used in a transparent overview layer	113

4.42	Excentric Labelling	113
4.43	Larger Excentric Labelling example	114
4.44	Construction of a space-scale diagram	116
4.45	Basic trajectories in a space-scale diagram	117
4.46	Semantic zooming in a space-scale diagram	117
4.47	Displaying and controlling information density	121
4.48	Display and control of information density	121
4.49	Restaurant Finder	122
4.50	Macroscopic as a Finder	124
4.51	Taxonomy of presentation techniques for large graphical data spaces	127
4.52	Data State Model applied to web sites	131
4.53	Types of represented information	132
4.54	Direct/indirect manipulation	133
5.1	Toplevel hierarchy tree	145
5.2	Hierarchy tree	146
5.3	Lost after zooming	150
5.4	Constructing the context layer	151
5.5	Interacting with a context layer	153
5.6	Controlling the context layer	156
5.7	Gesture used to create Figure 5.5	156
5.8	Construction of the history layer	159
5.9	Controlling the history layer	160
6.1	Control Menus in Zomit	162
6.2	Portal sub-menu in Zomit	163
6.3	Components of Zomit	165
6.4	Zomit virtual world	166
6.5	Exchanges between the server and client	171
6.6	Seeheim model	172
6.7	Arch model	173
6.8	MVC model	174
6.9	PAC model	175
6.10	Zomit modelled by Arch	176
6.11	Zomit client modelled by PAC	177
6.12	Threads in the Zomit server	179
6.13	Threads in the Zomit client	181
6.14	Redrawing after a mouse movement during a pan	182
7.1	Components of ZoomMap	191
7.2	ZoomMap: zooming from genome to bands	193

7.3	ZoomMap: zooming from maps to sequences	194
7.4	ZoomMap: cytogenetic representation into genetic maps	195
7.5	ZoomMap: zooming on indices	197
7.6	Global view of the library's shelves	199
7.7	One of the library's shelves	200
7.8	Some of images of the library's books	200
9.1	Un pie menu et ses zones actives	208
9.2	L'utilisation d'un marking menu	209
9.3	Modifier l'échelle dans Acrobat Reader 3	211
9.4	Un Control Menu (sur la vue de la Figure 9.16a)	213
9.5	Un Control Menu sur deux types d'objet	213
9.6	Zoomer avec un Control Menu	214
9.7	Défilement avec un Control Menu	215
9.8	Sous-menu des lentilles dans notre interface zoomable	216
9.9	Degré d'indirection	217
9.10	Le Control Menu dans vreng	219
9.11	Sélectionner du texte avec un Control Menu	220
9.12	Échelles fixes dans un FlowMenu	221
9.13	Valeur variable dans un FlowMenu	222
9.14	Vue fisheye d'un programme C	225
9.15	Table Lens	226
9.16	Vues de ZoomMap et de sa couche historique	228
9.17	Construction de la couche de contexte	233
9.18	Notre interface zoomable avec la hiérarchie sur la droite	237
9.19	Implémentation client/serveur de Zomit	240
9.20	Vue globale de la bibliothèque virtuelle	243
9.21	Une vue des rayons de la bibliothèque virtuelle	243
9.22	Une partie les livres de la bibliothèque virtuelle	244

Chapter 1

Introduction

The size of the world's databases has been growing exponentially for many years and all the signs indicate that this process will continue for the foreseeable future. It is not sufficient however to just store data in databases to understand this information. Bytes in databases are not knowledge. To convert databases into knowledge, users need to be able to interact with and understand the contents of the databases.

The huge volumes of data, their complexity and diversity, and the large numbers of links between data items mean that most users cannot interact with their databases without the use of sophisticated visualization tools. These tools need to allow users to gain a global understanding of an information space and then allow them to concentrate on a region of the space that is interesting for a given task. Even when users are focusing on a small region of a large information space they need to be aware of the position of that region in the entire space. This awareness must include a vision of the long distance relationships that link the region currently being examined with other parts of the database.

Databases are typically no longer stored on the computer running the user's visualization program. These programs thus now need to take into account the fact that many users will be accessing databases over long distance network connections. Consideration must be given to where and how the different tasks in the visualization process should best be carried out.

1.1 Improving Interaction

Complex visualization techniques require continuous control from users. Users have to specify what part of a very large information space they want to see and how this region should be presented. These are not static choices but rather require a fluid control of an interface that uses change and movement as visual-

ization techniques as much as it uses fixed presentations. Standard techniques or iterators have not greatly changed since the development of the WIMP (Window, Icon, Menu and Pointer device) interface and the invention of direct manipulation. These interactors were designed for single independent modifications of an essentially static interface and are thus unsuited to frequent or continuous control of an interface.

One type of iterator used today allows users to choose an operation from a list of operations proposed by the system. These iterators, which are most commonly menus, are of various different types. More recent menu types were designed so that novices and experts use the menu in almost exactly the same way. This aids novice users to become expert users.

Different iterators allow users to control the operation that they have chosen. These tools, such as scroll bars and dialog boxes, are used after the selection of an operation but are independent of the selection interactor. This separation of the selection of operations from their control means that users have to move backwards and forwards between the selection interactor and the control interactor as they direct their movement through an information space.

Part I of this thesis discusses some of the types of menus used in commercial software and also some more recent research projects. We then describe our new menu, called a *Control Menu*, and how it relates to and fills a gap in current menu techniques. A Control Menu combines the selection of commands and their control into a single interactor. This interactor, even though it combines these two functions, still only uses a mouse and a single mouse button. This integrated interactor provides the rapid and fluid control that is required by complex information space visualization systems.

1.2 Adding Context

Many different visualization systems for large information spaces have been proposed by research papers; some of these systems have been used in successful commercial products. We concentrate on visualization systems that provide a view of the information space and with which users can interact so as to find the information that they are seeking. These systems, a number of which are presented in Part II of this document, can be classified into several different types. All these systems are confronted with the same problem: how to maintain a visible representation of the context of detailed views. If there is insufficient context, users become disorientated; even after only a short period of navigation, they no longer know where they are in the information space nor where to find the information they are looking for. They are “lost in hyperspace”.

Zoomable User Interfaces (ZUIs) are one promising type of visualization sys-

tem for large information spaces. ZUIs use semantic zooming to create a multi-dimensional virtual world where users can find and transform the information in databases. Interacting with these interfaces involves the frequent use of a number of commands. The two most important are zoom and pan. We have developed a ZUI that uses our new Control Menu and thus integrates the selection of these commands and control of the zoom or the pan.

These interfaces suffer from a lack of context. After having navigated in the information space and thus having left the original global view of the space, users are shown a view of the virtual world that does not contain any context information. As with the other types of visualization systems, users rapidly become lost.

We use space- and depth-multiplexed displays to add the missing context to Zoomable User Interfaces. Space multiplexed displays allow the same information to be shown with two different representations. In our ZUI we show a zoomable view of the information space and a hierarchical representation of the same space in two highly coupled parallel windows. The hierarchical representation of the space maintains a permanent view of the structure of the information space and the coupling between the two representations shows users their position in that structure as they navigate through the space.

We also developed two other context aids that use transparent depth multiplexed views and our new interaction technique, Control Menus, to provide temporary context views. These aids are created by user request and exist only during the gesture used to create and control them. They require the fluid and continuous interaction provided by Control Menus as their usefulness comes from their reactivity. The movement that results from this continuous control is also important in aiding users to separate mentally the transparent aid from the still visible view of the focus.

The second part of the thesis includes a description of some information visualization techniques, including Zoomable User Interfaces, followed by a presentation of our new context aids.

Part I

Interactors

Chapter 2

Today's Menus and Controlling Operations

Interacting with objects using computer interfaces requires selecting operations to be performed on these objects and then controlling these operations. In most user interfaces these two phases of the same interaction are separated into two distinct interactors. These two types of interactor are used in very different ways. This extra complexity slows down the interaction with the interface as actions are performed in two steps. It also makes the action less fluid due to the multiple changes in the user's focus of attention. This problem can be compounded by an initial preliminary step necessary to select the target object or objects.

In this chapter we discuss how operations are chosen, briefly how they can be then controlled, and the problems caused by the use of multiple interactors to perform a single operation.

We describe some types of menus as being “contextual”. By this we mean that users can indicate, as they use the menu, the object on which they want the action chosen from the menu to act. We ignore as non-contextual any menu in which the target object must be selected before using the menu.

2.1 Existing Menus

Menus are one method for selecting from a normally small set of possible choices. These choices are actions to be performed by an application.

Users are accustomed to standard menus such as pull-down menus, drop-down menus, option menus and pop-up menus. Research projects have proposed other types of menus such as pie menus, marking menus, etc. These menus, even though some of them were first proposed many years ago, are not yet widely known or

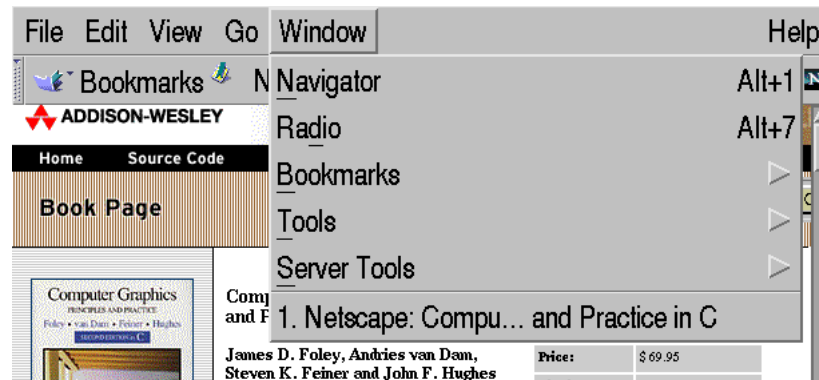


Figure 2.1: Pull-down menu in Netscape

used. In this chapter we list some of the menus used in standard applications and some other menus that are still research projects.

These menus have different characteristics which means that certain of these menus are more adapted than others to certain tasks. A taxonomy is proposed and used to contrast and compare the different menus.

2.1.1 Pull-Down and Drop-Down Menus

Pull-down and drop-down menus are grouped in menu bars. These menu bars are arranged along an edge of either the screen or an application.

If arranged along the edge of the screen, the menu bar is linked to the current application. The names and contents of the pull-down menus in the menu bar depend on the currently active application, and the functions chosen from a menu are applied to this application. This type of menu is used in the Macintosh user interface.

When a menu bar is in the same window as the application it is visually and logically linked to the application. An example of this type of menu is shown in Figure 2.1.

Using a pull-down or drop-down menu requires users to move to the menu-bar and thus cannot be contextual. The object or position on which the action selected in the menu must operate needs to be chosen before or after the use of the menu. The movement to the menu-bar becomes more and more penalising as application windows grow as users use bigger screens.

The user has to click on the corresponding entry in the menu bar, “Window” in Figure 2.1, to cause a pull-down menu to be opened. A drop-down menu opens, and is thus usable, as soon as the user moves the cursor over the corresponding entry in the menu bar. This automatic opening of menus can be distracting if

users are just moving the cursor from one part of the screen to another, especially to novice users.

2.1.2 Pop-up Menus

Pop-up menus differ from pull-down menus in that they can appear anywhere in the user interface. They are not limited to being connected to a menu bar. They are not normally associated with a visible user interface object that users can click on to make the menu appear. Users have to know the special operation that creates the menu. This special operation is most often pressing on a mouse button. The menu appears and is then used in one of two ways depending on the implementation of the menu. The first possibility is that the user continues to hold the mouse button down until the cursor has been moved over the desired item in the menu. This item is then selected by releasing the mouse button. The second possibility is that the menu is created by pressing and releasing the mouse button. Users then click on the required menu item or outside the menu to dismiss it without selecting an action. The first possibility means that the menu can be used in a single mouse movement but that the button must be held down while the user is deciding which item to choose. The second possibility means that the keyboard can be used to select an item in the menu. Using the keyboard is probably slower than continuing with the mouse.

The action of selecting an operation from a pop-up menu can be decomposed into three sub-tasks: invoking the menu, moving to the desired operation, and selecting the operation (Buxton, 1986). These sub-tasks are tied together into a single operation by the tension (user action) of holding the mouse button down. A pop-up menu is thus modal as during its use mouse movements have a special meaning. The duration of the modal state is however limited to the period when the user is pressing the button. It is the physical action of keeping the button down that creates the mode. Interactors where the mode is linked to physical actions, called *quasimodal interactors* by Raskin, are less error prone than normal modal interactors.

2.1.3 Pie Menus

The principal difference between a pie menu (Hopkins, 1991; Callahan et al., 1988) and a pop-up menu is that the menu items in a pie menu are distributed in a circle around the centre of the menu (Figure 2.2). Pie menus are thus radial rather than linear like standard menus. Users do not have to select a line in a linear list but can just move the cursor in the direction of the desired menu item and release the mouse button. A user who knows the position in the menu of the desired operation does not have to see the menu before making the correct

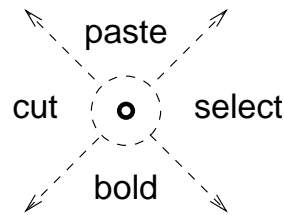


Figure 2.2: Pie menu and its active regions

gesture. No menu is shown to users who do not pause between the moment when they press on the mouse button and when they move the mouse. Pie menus thus adapt automatically and transparently to the presence of an expert user.

Pie menus are used in the game *The Sims* (<http://www.thesims.com>) to fluidly and rapidly control the game's characters (Macedonia, 2000). As pie menus are contextual they allow the action chosen from the menu to be associated with a character in the game without having to first select the character. Expert players can use these menus rapidly because they do not have to wait until the menu is drawn, nor look at it, before selecting an action.

A circular menu has also been proposed, as one of a long list in interaction possibilities, by the Logitech trackball in Microsoft Windows. Users click with the second button (the wheel) to display the circular menu. They can then use the trackball to navigate within the menu. A second mouse click is required to select the desired operation.

2.1.4 Marking Menus

Marking menus are similar to pie menus except that a line is drawn on the screen after each use of the menu. This line indicates the optimum gesture that the user could have made to activate the function that was chosen. This feedback tells novice users what gesture they could have made without seeing the menu and reassures the user who made a selection without the menu that the correct action was selected.

Kurtenbach and Buxton (1993) discuss how quickly a *marking menu* can be used by an expert. Marking menus are radial (or pie) shaped menus that pop-up with a press of the mouse. Menu selections can be made by either pressing a button for a short period of time (1/3 of a second) to pop up the menu and then by moving the mouse over an item and releasing the button to select it, or by pressing the button and moving the mouse in the direction of the (invisible) menu item and releasing the button (Figure 2.3). This second method of using a menu (marking) is used by experts. The advantage of marking menus is that the transition from novice to expert usage is simple; the expert's gesture is the

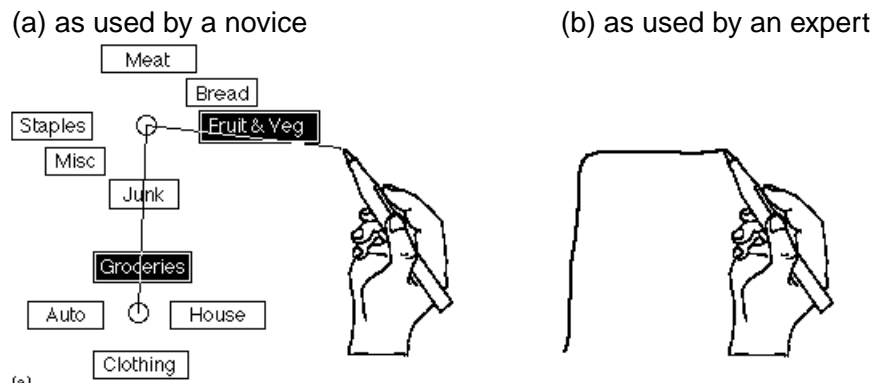


Figure 2.3: Marking menu use (adapted from Kurtenbach and Buxton, 1993)

same, except without the pause, as that of the novice. An intermediate method is also available. In this case the intermediate user makes the expert gesture without the menus but then waits until the menus are shown to get confirmation before releasing the mouse button. This is very different from traditional systems of menus and menu by-passes using an accelerator key. There the expert's gesture is completely different from that of the novice. Two different protocols have to be learnt.

Marking menus can have sub-menus. In this case the sub-menu pops up when the user moves the mouse past the appropriate menu item and pauses. An expert can use multiple menus without having to wait until they pop up, or wait for just one of the two menus to pop up.

Kurtenbach and Buxton (1994) describe a study where an existing program was modified to use a simple one-level marking menu. Their results showed that marking was used more than the menus, a user's skill with marking increases with use, and that marking menus can speed up the completion of a task by 10%.

Kurtenbach et al. (1999) propose the "Hotbox" (Figure 2.4), which is a new GUI widget that allows access to 1200 menu items. The Hotbox is popped up by holding down the space bar with the non-dominant hand. Each item is associated with a pull-down menu and each of the five zones shown (centre, left, right, top and bottom) has up to three marking menus (one on each button). The marking menus and the lines of items shown can be configured by the user.

2.1.5 Command Compass

The Command Compass from Momenta is described in Kurtenbach (1993). This menu was developed for a pen based computer system that was commercially unsuccessful (Reinhardt, 1991). The Command Compass is a circular menu like a pie menu. It differs however because actions are not selected by lifting the pen

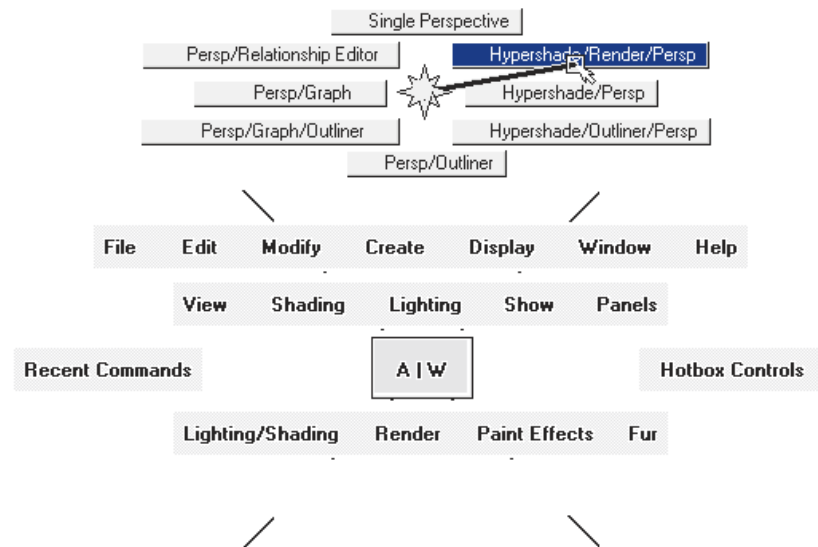


Figure 2.4: Hotbox (adapted from Alias|Wavefront, 2000)

but rather by simply moving the pen far enough into the compass sector associated with the desired action. Subsequent pen movements are used to control the chosen action. An example is a move operation. The pen touches the surface over the object to be moved. The move operation is selected by moving the pen in the direction of that action in the circular menu. Once the pen is in the compass sector, the object starts following the cursor. The new final position of the object is indicated by lifting the pen.

The Command Compass always draws the menu but this can be ignored by the expert user. Only one level of menu is possible and no marks are drawn to indicate what operation was chosen. The lack of marks was probably not a problem for users given the simplicity of the menu.

2.1.6 FlowMenus

FlowMenus (Guimbretière and Winograd, 2000) are an extension of pie menus and marking menus. They were however designed for pen based interfaces. The essential difference is that the choice of an operation is indicated by returning the cursor to the centre of the menu. This allows a number of different operations to be performed sequentially without lifting the pen. Another possibility is the control of the chosen operation. This control can be effected by the use of a further menu that allows parameters to be selected or by direct manipulation. Figure 2.5 shows the pen movements (that are not normally visible) when using a FlowMenu to

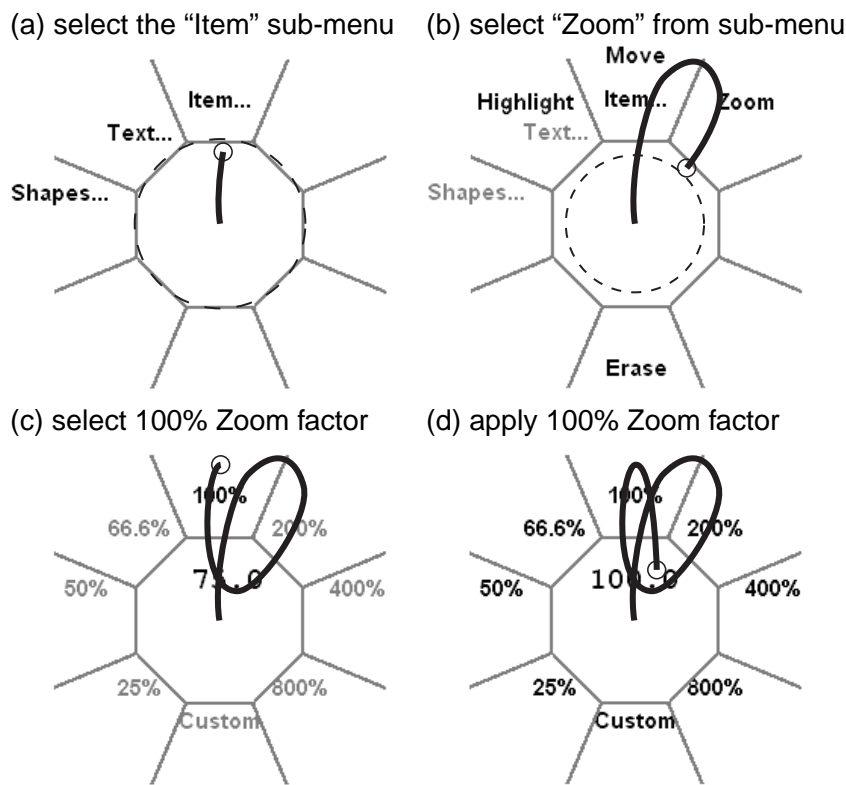


Figure 2.5: Fixed zoom factors in a FlowMenu (adapted from Guimbretière and Winograd, 2000)

select the Zoom command from a sub-menu and then the choice of the 100% zoom factor. At this point the pen is still on the surface so the user can try a different zoom factor by moving to another quadrant and back to the centre. This method of selecting a zoom factor only allows preset zoom factors to be chosen and is equivalent to choosing from a sub-menu. A knob interaction method can be used, giving a large number of zoom values in predefined increments. In Figure 2.6, the user has selected the custom zoom factor. From this moment, until the pen is removed from the surface, clockwise movements of the pen around the centre of the menu increase the scale (by a fixed amount) each time a segment boundary is crossed. Similarly, anticlockwise movements decrease the scale. Throughout this interaction the user has to concentrate on the menu, as the segments are not very large, and on the effect of the interaction on the objects in the interface. These objects are visible under, and partly hidden by, the transparent FlowMenu.

A FlowMenu can also be used to directly manipulate objects in the interface. Moving an object is one possible example. The menu is contextual so the object to be manipulated is chosen by the position of the pen when it touches the surface

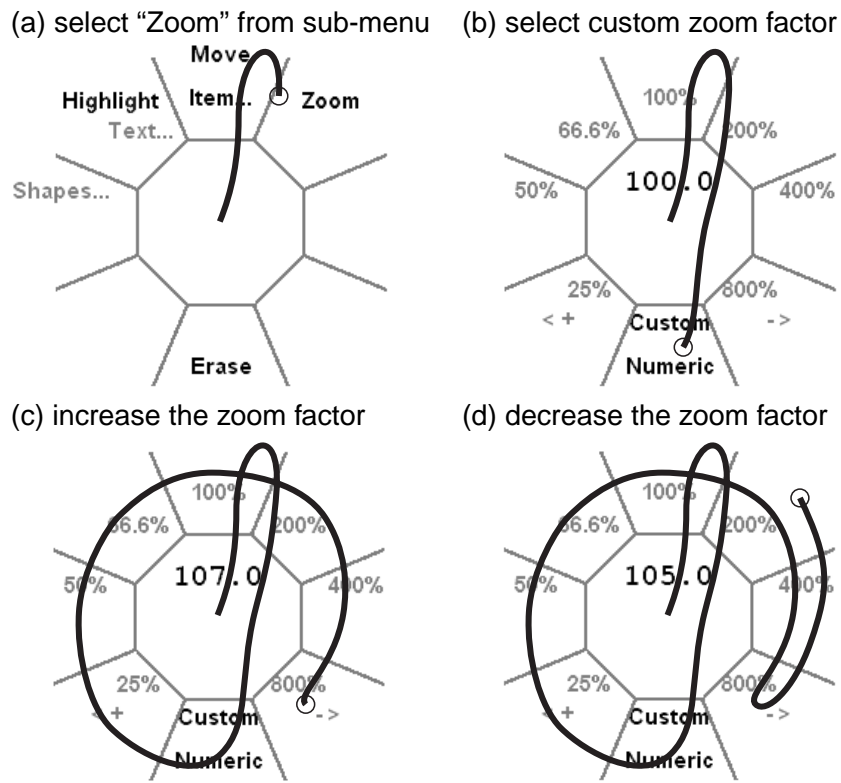


Figure 2.6: Variable zoom factor in a FlowMenu (adapted from Guimbretière and Winograd, 2000)

and pops up the menu. An action that involves direct manipulation is chosen in the same way as any other command. If the action is a move, then as soon the command is selected (by the cursor returning to the centre of the menu) the object jumps to its new position under the cursor and then follows the cursor until the pen is lifted from the surface.

An operation in a FlowMenu can also require the user to enter some text. This text entry is performed using the Quikwriting system (Perlin, 1998) which provides a way to enter a number of characters in a single gesture.

FlowMenus were developed after our Control Menu (presented in chapter 3 of this thesis).

2.1.7 Fisheye Menus

Fisheye menus (Bederson, 2000) apply fisheye visualization techniques (Furnas, 1986) to linear menus. They are designed to facilitate the selection of an item from a list that is too long to fit on the user's screen. Fisheye menus change the size of

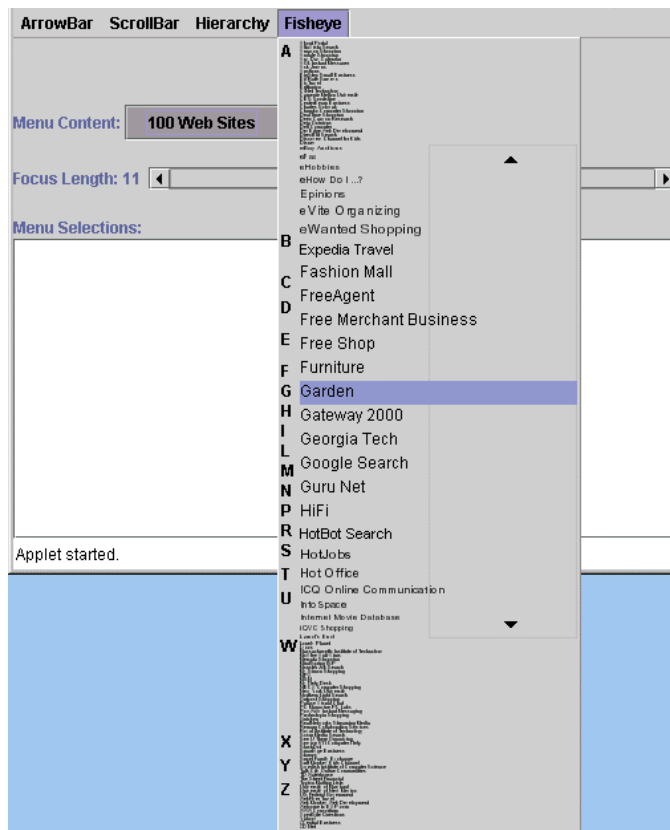


Figure 2.7: 100 item fisheye menu (adapted from Bederson, 2000)

the font used to draw the items in the list in such a way that all the items in the list remain visible but not necessarily readable (Figure 2.7). The user has a current position in the list. The items in the list at and around this position are drawn large enough to be easily readable. These are the focal items. The other items (the non-focal items) are drawn sufficiently small so that they fit in the remaining vertical screen space. As the user moves the cursor, and thus the current position in the list, the position of the focus changes; previously readable items become too small to be readable and previously unreadable items, in the direction of the user's movement, become readable.

As not all items are readable, the list items must be ordered to avoid users having to scroll slowly through the entire list. It is not possible to use a fisheye menu with unordered data nor to change the order of the items to group items by function type. This order is normally alphabetic and fisheye menus have an alphabetic index. This index shows, in a readable font, the letters of the alphabet for which there is at least one item that starts with this letter. Moving the cursor to the position of a letter makes the first item in the list starting with this letter

readable. These menus do not require the use of scrollbars and do not unnaturally impose a hierarchical organisation.

Figure 2.7 shows a fisheye menu being used to select the name of a web site from a list of 100 names.

2.2 Taxonomy

A number of publications (Kurtenbach, 1993; Foley et al., 1996) propose criteria that can be used to classify different menu designs. This section uses those criteria to classify the different menus already presented.

2.2.1 Menu Placement

The position of menus determines the time required to find a menu and to move the cursor to it. Menus can be placed in the following ways:

on keyboard or tablet Some menus are always available and in a fixed position because they are printed on keys on the keyboard. These menus are always to hand and are constant across applications; it is however impossible to customise them for a particular application and applications need a way to find out what menu keys are available on a given keyboard. The Xerox Star used these menus (Figure 2.8) and Sun keyboards still include menu keys.

Other menus might be printed on the edges or even on the surface of a graphics tablet. Again the functions are always available but users must release the mouse and move to the keyboard to use these functions. Also, applications cannot add to or change these menus and different keyboards have different menu keys.

fixed screen position The Apple Macintosh user interface imposes a single menu bar at the top of the screen shared by all applications. Only the current applications menu is visible and the application is limited to the format imposed by the system. Customisation of the menu bar by a given application is limited. The contents can be changed but two line menu bars, for example, are not possible.

with the application Other than on the Apple Macintosh, pull-down and pull-out menus, are found in menu bars attached to the application.

user positioned A tear-off menu is a pull-down menu that has been disconnected from its menu bar and that can be placed by the user at the desired position for a given task. Figure 2.9 shows four tear-off menus from Sun's OpenWindows mailtool that have been positioned above the application.

(a) left cluster of function keys



(b) right cluster of function keys



Figure 2.8: Xerox Star function keys

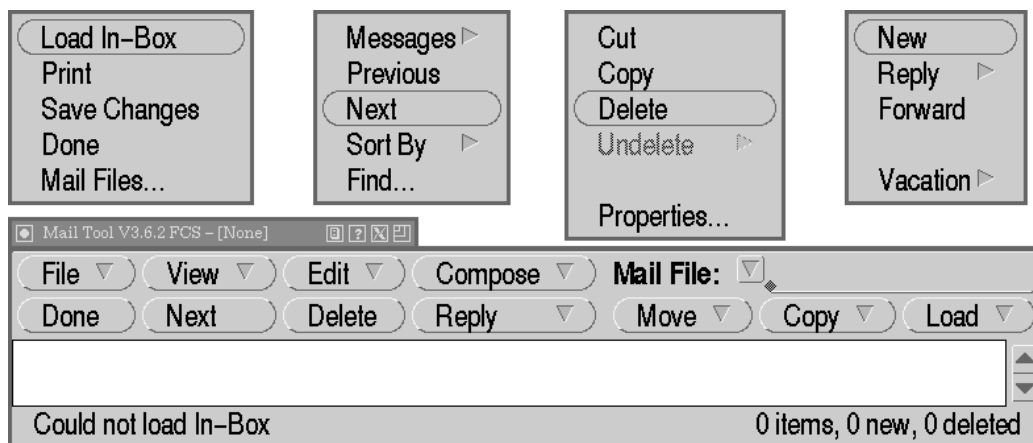


Figure 2.9: Tear-off menus in Sun's mailtool

under the cursor All the different pop-up menus appear directly under the cursor. However, a problem arises when the cursor is too close to the edge of the screen for the menu to be drawn in its normal position relative to the cursor. Three solutions are possible: warping (moving) the cursor away from the edge of the screen, clipping the menu, or drawing the menu in a different position. Warping the cursor is often distracting for users and is confusing when the menu is contextual as the position of the cursor is important. Clipping the menu means that some of the menu items are invisible and thus impossible (or at least difficult) to use. Drawing the menu at a different position invalidates the movements users have learnt to activate menu items.

mouse buttons Chords can be used to execute a number of different commands. This can involve giving multiple mouse clicks a different meaning from a single click or assigning a meaning to a second button being pressed while another is being held down. Mouse wheels are another way of controlling applications with additional input devices.

full-screen Some menus are sufficiently important that the menu should block the entire screen. This is most often used for the menu which asks “do you really want to logout?”

2.2.2 Visual Representation

The elements in menus can be depicted with textual names, icons or other graphical representations. Textual names are almost always long and thin while icons and graphics can be designed in many different shapes that are often more compact than textual strings (Figure 2.10). Icons are especially compact when they concern graphical commands as in the example. Good icon design is an art. Icons thus give more flexibility in menu design which is particularly important when using non-linear menus. The pie menu in Figure 2.11 from the game SimCity (Perkins, 1993), available at <http://www.art.net/~hopkins/Don/simcity>, shows how icons can be distributed in a circle.

Menus can also be transparent. Transparent menus have the advantage of hiding less of the user's workspace than solid menus. Their disadvantage is that they can be more difficult to read.

2.2.3 Size and Shape of Menu Items

Fitts' Law (Fitts, 1954; Fitts and Peterson, 1964) says that larger items are easier to select and so large menu items should be easier to use. Smaller menu items












	circle: specify radius
	circle: specify diameter
	ellipse: specify radii
	ellipse: specify diameters
	closed approximated spline
	approximated spline
	closed interpolated spline
	interpolated spline
	polygon
	polyline
	rectangular box
	rectangular box with rounded corners

Figure 2.10: Xfig buttons and corresponding text

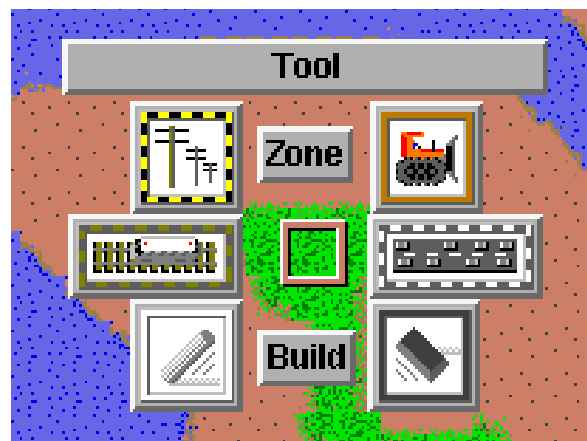


Figure 2.11: Pie menu in SimCity

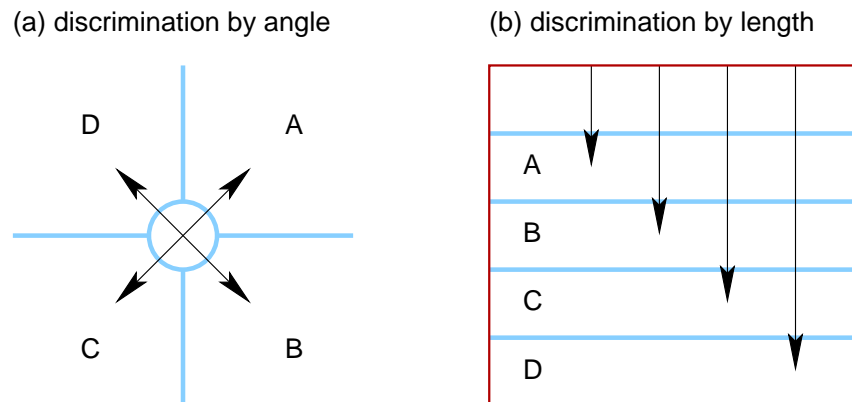


Figure 2.12: Menu item selection discrimination

take less space and allow more items to be displayed in a given area but increase the time taken to select items and increase the number of selection errors.

Circular menus (Figure 2.11) are drawn with the cursor at the centre of the menu; users move the cursor in the direction of the desired operation. The further the user moves from the centre of the menu, the larger the target, and thus the smaller the likelihood of error. Users control the speed versus error rate tradeoff.

2.2.4 Menu Shape

Regardless of the shape of menu items, menus are either linear or circular. The shape of the menu defines the discrimination method of the menu. This can be either by angle or by length as shown in Figure 2.12.

linear Linear menus require users to watch the menu as they move the cursor to the required line.

circular Selecting an item from a circular menu requires a movement in the correct direction. If the menu does not contain too many items the direction of frequently used items can be learnt and users no longer need to watch the menu when selecting item.

Circular menus have another advantage: the length of the mouse movement required to select each menu item is the same (Figure 2.12). Any item can be selected as rapidly as any other, and the position of items in the menu does not depend on the frequency of use of the items.

Choosing accurately amongst more than twelve items in a circular menu requires an excessively large mouse movement, and laying out a very large

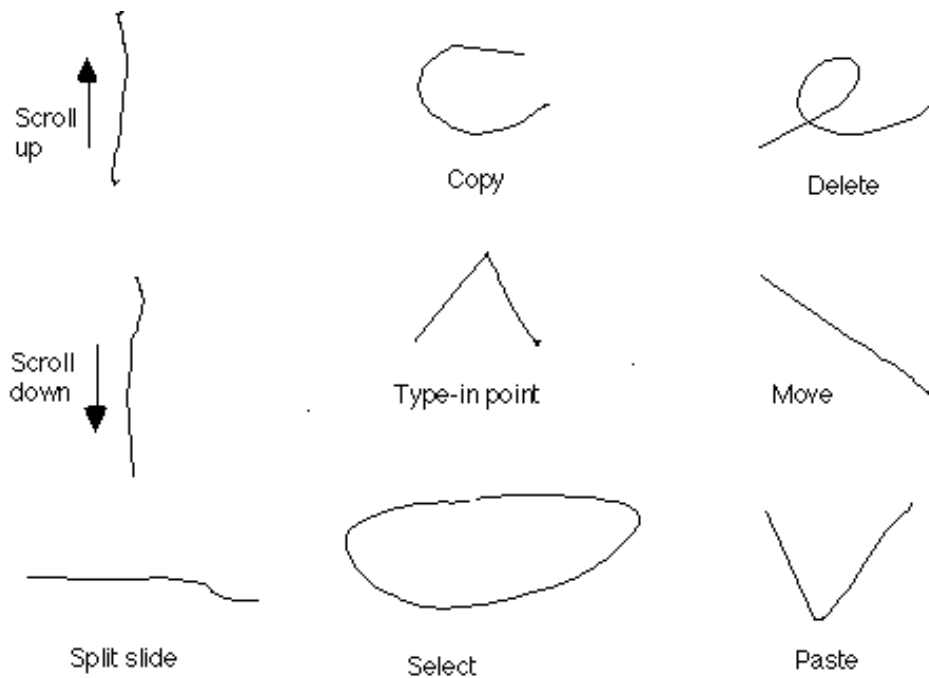


Figure 2.13: Gestures in Tivoli (adapted from Kurtenbach, 1993)

circular menu consumes a lot of screen space. Circular menus thus have the disadvantage of being limited in size.

2.2.5 Visible Versus Invisible Menus

Menus need not always be visible. Users can choose between actions using gestures (movements with a continuous-positioning device such as a tablet or mouse). The pattern recognition software compares the user's gestures with a predefined set of gestures and chooses the gesture that matches.

With this technique users do not have to move from their work area to a command input device (a keyboard for example) or to a menu bar. Tactile continuity is thus maintained.

Some patterns are derived from already existing gestures such as proofreader's marks. Others are defined for a particular computer system. Figure 2.13 shows the basic edit marks in the pen-based electronic whiteboard application called *Tivoli*. Others, such as those used in marking menus, are the same as the gesture made when using the menu. In this case, the patterns are learnt by the users as they use the menus (Figure 2.14).

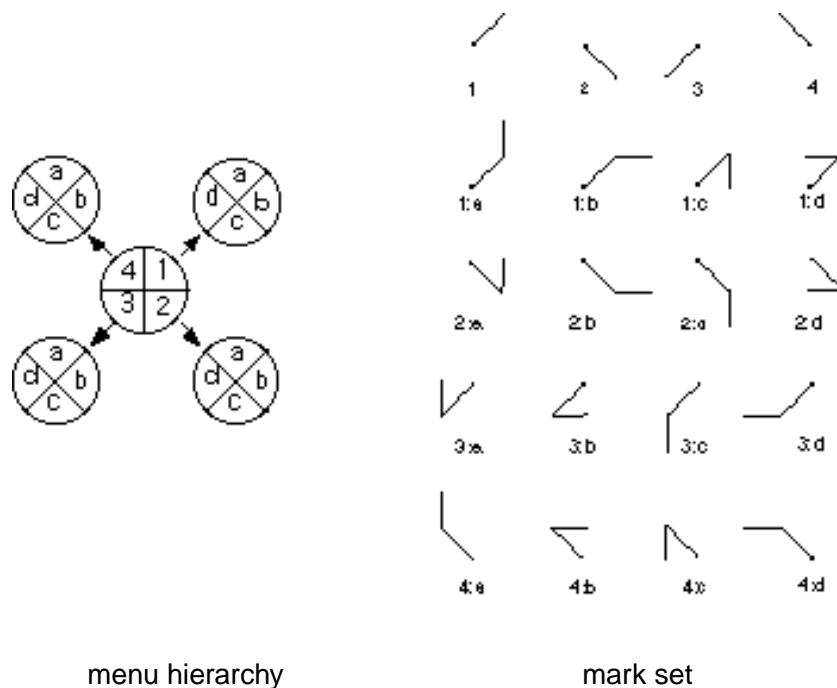


Figure 2.14: Circular menu hierarchy and the associated marks (adapted from Kurtenbach, 1993)

2.2.6 Interactor Distance

Users work on objects near the current position of the cursor. Any interactor that requires a large movement of the cursor from this position has a significant spatial offset. This is the case for menus in menu bars or in other fixed positions on the screen. Pop-up menus and gestures differ in that they are activated at any point in the user interface. This avoids the movement of the cursor from the current position and back.

Users not only use the mouse, they also use the keyboard. If users spend most of their time working with the mouse then it is reasonable to concentrate on the mouse. If users spend most of their time working with the keyboard then any use of the mouse will be disruptive as moving from the keyboard to the mouse and back again is a relatively long operation. In this case the use of function keys or keyboard accelerators is more appropriate.

2.2.7 Contextual

Contextual menus, that is menus that are activated anywhere in the application, adapt to the object currently under the cursor. This makes menus shorter as inap-

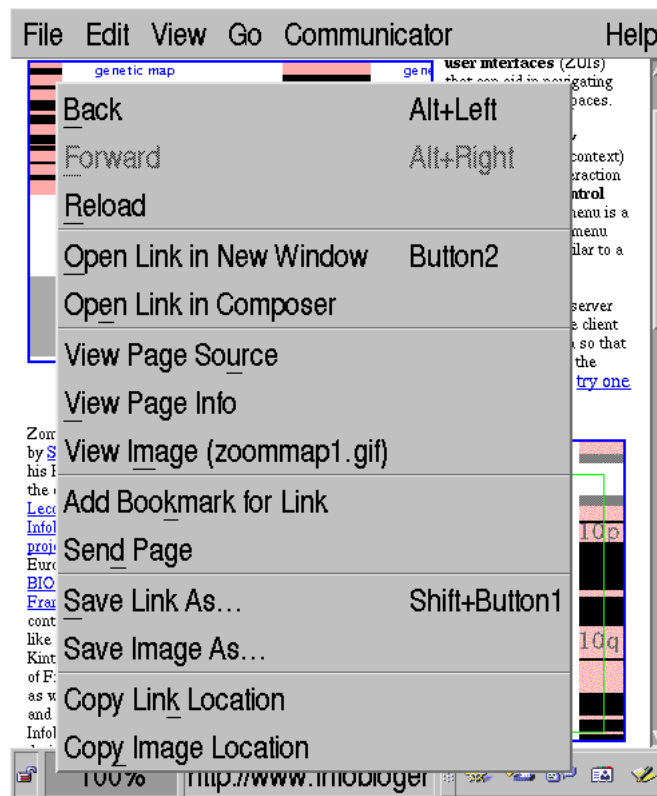


Figure 2.15: Contextual menu in Netscape Communicator

propriate functions can be removed from the menu.

Interactors (in general) that can be activated anywhere in the application, such as pop-up menus, are contextual in another sense as well. The action chosen from the interactor can be associated with the position of the cursor when the interactor was activated. The system creates a list of objects at this position. Once the action is chosen, those objects in the list that are appropriate for this action are arguments to the action.

The example in Figure 2.15 shows a contextual menu from Netscape Communicator. This menu is contextual in two different ways. The menu was activated over an image that is also a link. The menu thus contains commands, such as “View Image”, that are only appropriate for images and commands that are only appropriate for links, such as “Open Link in Composer”. If a command that applies to images is chosen it will be applied to the image that was under the cursor when the menu was activated. Similarly, if a command that applies to links is chosen it is applied to the image’s link. The pop-up menu also contains commands, such as “Back”, that are independent of where the menu was activated.

2.2.8 Keyboard Accelerators

Many menu systems provide keyboard accelerators. These accelerators can be modeless; the key stroke sequence can be used at any moment when the user is not in another mode. In this case the accelerator normally consists of the use of one or more modifier keys and another key. These accelerators are often displayed in the menu that provides the equivalent function. Figure 2.1 shows a menu where two keyboard accelerators for the first two functions in the menu, “Navigator” and “Radio”, are indicated to on the right hand side of the menu. The keyboard accelerator, pressing “1” while the “Alt” modifier is pressed, can be used at any time to change windows.

Other accelerators can only be used when the menu that contains the desired function is already shown. As these accelerators are modal and associated with the given menu, they can be simpler and are often a single keystroke. In the menu in Figure 2.1 the keys of the modal keyboard accelerators are underlined in the text or the menu item. When, and only when, this menu is open, the “t” key can be used to open the “Tools” sub-menu.

These accelerators are very common but many users do not use them. Some users have probably not yet discovered them but others have decided not to use them. Bederson (2000) states that this could be because their hands are already on the mouse and recentring their hands on the keyboard and finding the right key would take too long. Others might not know the exact name, let alone the correct keyboard accelerator, of the function they want. Yet others might not like using the keyboard when they are using menus. Also, the internationalisation of these menus is difficult as the letter used in the accelerator must be in the text of the item. When the language changes it is often necessary to change the accelerators as well.

2.2.9 Self-Revealing

Interactors are self-revealing if users do not have to be shown where to find them. One class of self-revealing interactors is the buttons that users just have to click on to activate a function. An example, shown in Figure 2.16, is the most important commands in Netscape (“Back”, “Reload”, “Home”, etc).

Even more self-revealing are the keyboard menus of the Xerox Star. They are even too self-revealing in that the developer of an application cannot hide an element of a keyboard menu even if it is not applicable in that application.

Pull-down menus are less self-revealing as only the name of the menu is displayed. Looking at the Netscape menus shown in Figure 2.16, users have no way of knowing that the “Search Internet” function is available on the “Edit” menu. If users do not know on which menu a given function is located, they have to search

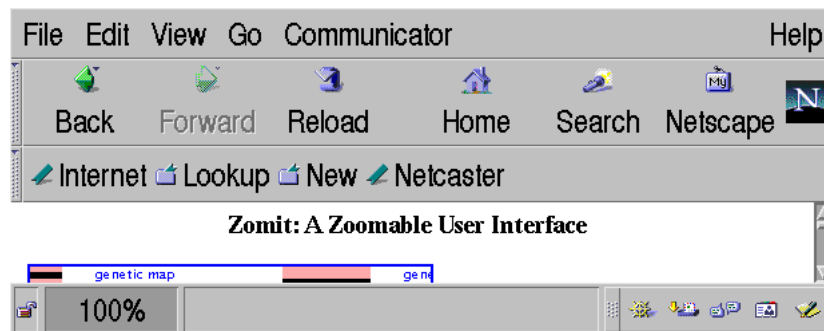


Figure 2.16: Buttons in Netscape Communicator

on all the available menus and sub-menus.

Pop-up menus are not self-revealing because there is no indication in a user interface that they even exist. Users can only press the mouse button that is normally associated with contextual menus to see if there is a menu and to find out what it contains. Pop-up menus whose contents (and not just the active items) depend on where the menu is activated (depending, for example, on what type of object is under the cursor at the moment of activation) are even more difficult to find. Users have to experiment with the mouse button over each type of object to find all the possible menus.

2.2.10 Novice and Expert Gestures Similar

Many systems provide two or more ways of activating the same function. The most obvious, but slowest, way of activating a function is designed for novice users. If, having already learnt the first method of selecting functions, they feel the need to work more rapidly then they need to learn the other, less obvious, method of selecting the same functions. This second learning step is facilitated if the faster activation method is indicated by the novice method and greatly facilitated if the two methods are similar.

The contextual menu shown in Figure 2.15 indicates that the “Back” operation can be activated by hitting the “left” key while holding down the “alt” modifier key. The novice use of the pop-up menu thus indicates how to use the more expert method but the two methods are completely different.

Marking menus (subsection 2.1.4) are designed so that the novice’s use of a menu is very similar to that of an expert. Novice users thus become almost automatically expert users.

2.2.11 Combines Selection and Control

Most menus only allow a choice to be made between a small number of options. If the choice is a command that requires parameters, a different interactor must be used after the menu. These other interactors can be dialog or property boxes. The use of dialog boxes is modal since the interface is blocked until the new values are entered into dialog box and the box dismissed. It is thus impossible to use the interface to find the values to enter into the dialog box as the dialog box blocks the interface. Modal interfaces are generally recognised as being error-prone and difficult to understand (Raskin, 2000).

FlowMenus (and the short-lived Command Compass) allow parameters to be supplied to the chosen function. This interactor is more than a simple menu as it includes control functions that are normally absent from menus; they are provided by other interactors such as scroll bars and dialog boxes.

FlowMenus are modal as during its use mouse movements have a special meaning. The duration of the modal state is however limited to the period when the pen is touching the surface of the device. This menu is thus only quasimodal rather than modal.

2.2.12 Summary

Table 2.1 summarises the different properties and menus presented in this taxonomy.

2.3 Controlling Operations

As discussed in subsection 2.2.11, menus are generally only used to select an operation. Other interactors are required to complement menus if the operation requires parameters (other than that provided by the activation location where the menu is contextual).

Beaudouin-Lafon (2000) introduces a model for evaluating interactors according to their properties.

degree of indirection: a two dimensional measure of the *spatial* and *temporal* offsets of the interactor (Figure 2.17). The spatial offset is the screen distance between the interactor and the object it operates on. The temporal offset is the time difference between the user's manipulation of the interactor and the response of the object.

degree of integration: the ratio between the number of degrees of freedom provided by the instrument and the number used by the input device.

	Menu Type					
	pull-down	option	pop-up	pie	marking	flow
activation method is self-revealing	yes	yes	no	no	no	no
must move to activate	yes	yes	no	no	no	no
contextual	no	no	yes	yes	yes	yes
discrimination method	distance	distance	distance	distance	angle	angle & distance
maximum items per menu	screen height	screen height	screen height	8	8	8
selection confirmation	yes	yes	yes	yes	yes	
menu visible	yes	yes	yes	during or after use	after pause	transparent
expert rehearsal	no	no	no	yes	yes	n/a
“eyes free” use	no	no	no	yes	yes	no
mark after use	no	no	no	no	yes	no
sub-menu selection by	movement	none	movement	pause	angle	
final selection by	button-up or tap	button-up or tap	button-up	button-up	button-up	move to centre
control method	dragging or tapping	dragging or tapping	dragging or tapping	dragging	dragging	dragging
parameters supplied	no	no	no	no	no	yes
designed for	mouse	mouse	mouse	mouse	mouse	pen

Table 2.1: Menu summary

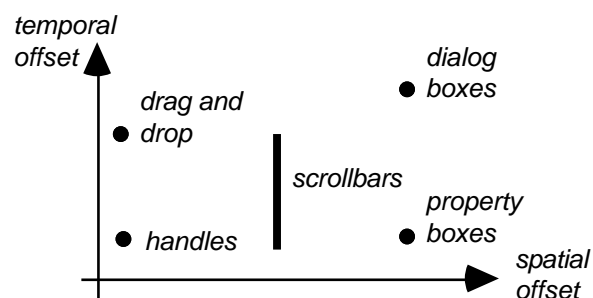


Figure 2.17: Degree of indirection of an interactor (adapted from Beaudouin-Lafon, 2000)

degree of compatibility: the similarity between the physical actions of the users on the instrument and the response of the domain object.

Beaudouin-Lafon (2000) applies the interaction model to standard WIMP interactors. New interaction techniques often found in WIMP interfaces, such as inspector, property boxes, drag and drop, and contextual menus, are shown to be more efficient than their standard WIMP counterparts. Post-WIMP interfaces are also analysed following the interaction model.

Navigation in zoomable user interfaces is fast because the navigation instruments (mouse buttons and modifier keys) have low temporal offsets. Their use also has high degrees of compatibility and integration. The design of a new text search instrument is presented and discussed in terms of the interaction model.

2.4 Problems With Existing Interactors

The problem with pop-up and marking menus is that they do not allow the chosen operation to be controlled interactively. They also do not allow any parameters to be supplied. For instance, an operation such as a font size change often requires a dialog box to supply the new font size. Users must use the menu and then switch their attention to another interactor. Once the new size has been entered the dialog box is dismissed and the users must refocus their attention on the workspace.

An action that should be considered by the user as a single action, or *chunk* in the terminology of Buxton (1986), has been decomposed into a number of actions. A novice user must understand that these actions are linked and all need to be executed to perform a single operation. Only when users have become experts, through repetition of the task, will they be able to consider these actions as a single chunk and thus ignore the low level details of multiple interactor use and concentrate on their high level task.

Pan operations require either a dedicated mouse button (or a mouse button and a modifier) so that users can drag the image, or two scroll bars. Panning cannot be done with standard menus except with impractical commands such as “move a little to the right”. Zooming is another operation that is difficult to perform with menus. Users want to zoom until the correct scale has been obtained. Standard menus only allow them to zoom in fixed steps, and then only via repeated uses of the menu.

2.5 Example: the Scale in Acrobat Reader

There are four different ways to change the scale in Acrobat Reader™ from Adobe: a pull-down menu (Figure 2.18a), an option menu (Figure 2.18b), a dialog box

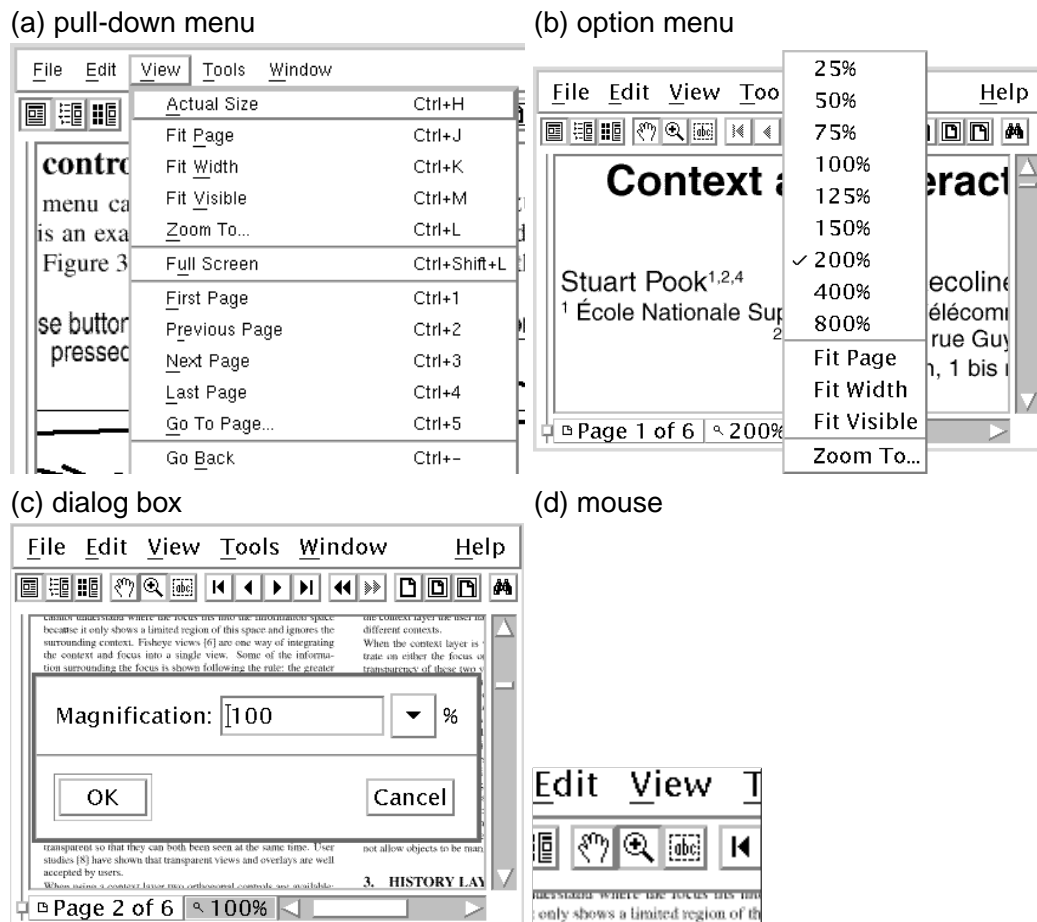


Figure 2.18: Changing the scale in Acrobat Reader 3

from a pull-down menu (Figure 2.18c), or via a mouse button. The use of the mouse button to change the scale of the view requires that the mouse be in the correct mode. The mouse button can be used to pan the view, zoom on a point (increase the scale), select text, or dezoom (decrease the scale). The current mouse mode is indicated by the cursor and three buttons shown in Figure 2.18d. The first three mouse modes can be selected with the buttons. The fourth mode (dezoom) must be selected using the “Tools” menu item (or a keyboard accelerator) in Acrobat Reader 3. With the fourth version, the dezoom mode can be selected via a menu that appears (rather unexpectedly) when the user presses on the button that normally selects the zoom mode (Figure 2.19).

All these four methods of changing the scale have disadvantages.

Pull-Down Menus The pull-down menu requires users to move the cursor away from their text to the menu bar, click on the button, focus their attention to

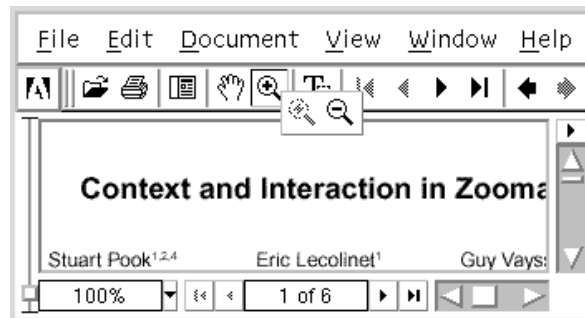


Figure 2.19: Selecting the dezoom mode in Acrobat Reader 4

the menu and select the correct item. Once the appropriate menu item has been chosen, the menu disappears and users have to refocus their attention on their text. This list of possible scales is also very limited (but the menu is already very long). The menu bar also uses valuable screen space. This loss (or waste) of space is particularly annoying here because menu bars are horizontal (thus using vertical space) in an application where vertical space is most valuable, since screens are landscape sized but documents are normally portrait sized.

Option Menus The option menu also requires a mouse movement from the user's text to the menu's button, then to the correct line in the menu, and then back to the user's text. The user's attention must follow these mouse movements. A certain amount of screen space is used by the option menu. This space is used to show the current scale which requires the user to learn that the scale can be changed by clicking on "200%". The choices offered are limited, and a user who is unhappy with a choice made must restart the whole process to choose another scale.

Dialog Boxes The dialog box allows an exact scale to the specified but using a dialog box is even slower than a pull-down menu or an option menu. One of these menus must be used to display the dialog box, the correct value must be entered, and then the dialog box dismissed. There are thus even more mouse movements and attention changes than with the menus; attention focus on the menu, then on the dialog box, and then back to the users text. If the new scale is not correct, and no help is provided to aid the user in choosing the correct scale, then the dialog box has to be redisplayed in order to enter another scale. No "apply" button is provided, so it is not possible to see the effect of a new scale without dismissing the dialog box.

Mouse Buttons To use the mouse button to change the scale, the program must

be in the correct mode. If the mouse button is in a different mode, the user must click on the appropriate button to choose the correct mode. This is an extra step and the buttons used to change the mode occupy valuable screen space. The usual way of changing the scale with the mouse is by clicking. This changes the scale by fixed steps. This makes it impossible to choose exactly the correct scale and if the desired scale is passed then program mode has to be changed, via the buttons (Figure 2.18d), to change the scale in the other direction. A second way to use the mouse to zoom involves sweeping out a rectangle over the document. This rectangle indicates the area the user wishes to see. The scale is then adjusted so that only that part of the document is visible.

None of these methods of changing the scale is perfect. Finding the correct scale is an iterative process, undoing a scale change is not easy, and, with the menus, too many focus changes and mouse movements are required.

Chapter 3

A New Interactor

The taxonomy of menus in the previous chapter shows that there is a need for a menu that combines the selection of operations and the control of the chosen operation. This new menu should be contextual or pop-up because such menus do not require large mouse movements to be activated and allow the target of the operation to be indicated by users.

We thus propose a new type of contextual pop-up menu (Figure 3.1) called a *Control Menu*. These menus resemble pie menus and combine the selection of an operation and the control of this operation. They integrate up to two scroll bars or spin-boxes and thus allow users to keep their attention focused on the menu during the operation. Control Menus can have sub-menus, and also retain the novice and expert modes of use found in marking menus. This menu not only allows operations to be selected, it also allows those operations to be controlled via the same interactor in a fluid and continuous way.

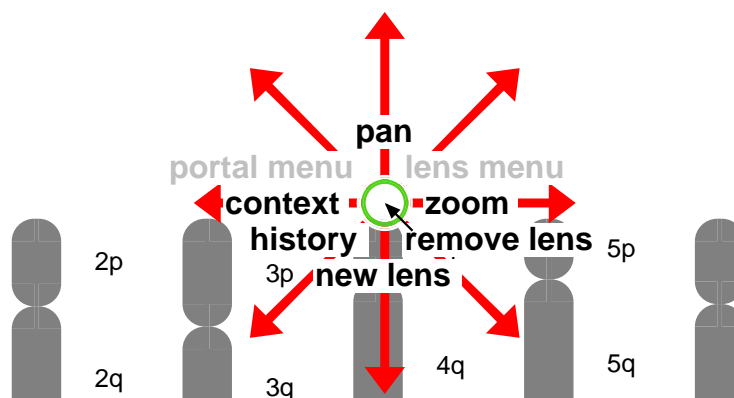


Figure 3.1: Control Menu in a Zoomable User Interface

3.1 Control Menu

In this section we describe the basic properties of Control Menus; those that are shared by other types of menus.

3.1.1 Pop-up Menu

A Control Menu can be activated anywhere in the user interface by simply pressing on a mouse button. Control Menus do not require users to move the mouse (or pen) to a visible interactor such as a button or a menu bar. Users can thus keep their attention focused on the centre object on which they are currently working and need not make large mouse movements to and from visible menus in fixed positions. As there are no visible indications that the menu exists, users need to be told that pressing a mouse button will activate a menu. This very common comportment of user interfaces is simple to explain and easy to remember.

It is possible to associate a different Control Menu with each mouse button or even to change the contents of the menu depending on the keyboard modifiers that are pressed when the mouse button is pressed. This technique is used by Hotbox (subsection 2.1.4). As even non-hierarchical Control Menus can contain eight items and hierarchical Control Menus many more, this possibility will be needed only by more complex user interfaces.

3.1.2 Contextual Menu

Control Menus are contextual. They can be activated anywhere on the user interface and the activation position can be used to indicate the point on which the user wishes to act. This position can also indicate one or more objects that are candidates to be used by the operation. The exact set of objects to be used by the operation is determined once the operation has been chosen. This allows a Control Menu to be activated over a set of objects of different types. Only those objects that are compatible with the chosen operation are taken into account by the operation. Figure 3.2 shows a Control Menu being used at a position where there is both a portal and a lens. Both the portal and lens sub-menus in the Control Menu are active. The system will use the choice of the operation to decide whether it is the portal or the lens that is to be the target of the operation.

The contents of the Control Menu can also be adapted to the position where it is activated (or the objects at this position). This avoids proposing operations that are not appropriate at the current position. It is however probably desirable to organise the positions of the items in the menu so that when they are active they are always in the same place.

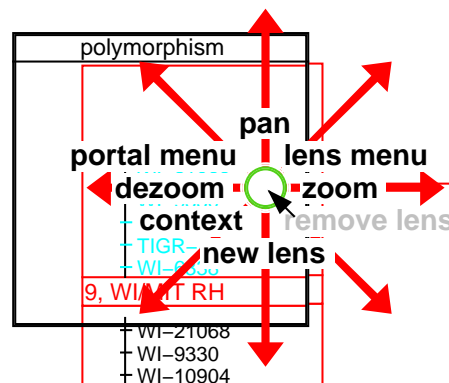


Figure 3.2: Control Menu and two types of object

3.1.3 Circular Menu

Circular menus, such as Control Menus, are easier to use than linear menus. All the targets in the menu are near and at the same distance from the centre of the menu. The further users move the cursor from the centre of the menu the larger the target and the lower the risk of error but the slower the selection. Users can thus choose the tradeoff between the risk of error and the speed of selection and this with each use of the menu.

Circular menus are “eyes-free”. Experiments (Kurtenbach, 1993) show that four, eight and twelve item menus are easy to use and that even when the menus are hidden. Bigger menus and menus with an odd number of items were found to be difficult to use. Users thus do not have to wait until a circular menu is shown before selecting an operation. This is not possible with linear menus. It is easy to move the cursor down and left, or example, from the centre of a circular menu but much more difficult to move down to say the fourth line in a linear menu.

Remembering compass quadrants, north, north-west, west, etc, appears to be easier than positions in a linear menu. The movement to select an operation from a Control Menu is a single action in a single direction that soon becomes automatic. Selecting a line in a linear menu requires eye-hand coordination as the user has to move down and to the right (depending on the exact design and position of the menu) and then stop in the correct target. This complex operation is harder to remember.

3.1.4 Menu Visible Only for Novices

Users press on a mouse button to activate a Control Menu. If the user does not move the mouse within a third of a second, the menu is shown. The novice user can then use the menu to select the desired operation. The menu is not shown

if the user knows the position of the desired operation and immediately moves moves the cursor in the direction of the operation.

A user who does not need the menu is not distracted by its appearance and the focus of the user's attention (that part of the workspace under the cursor) is not even temporarily or partially hidden from the user.

The choice of novice or expert use of the menu is automatic. Users need not decide if they are novices or not. There is no button to check to say whether one is a novice, and the choice of novice or expert mode is made automatically at each menu use. If the user hesitates in making a choice the menu is shown as a reminder, otherwise it is not needed and not shown.

3.1.5 Novice and Expert Modes Similar

Many menus provide an expert mode. With traditional menu systems the expert way to choose operations from a menu does not use the menu at all. Users have to learn keyboard accelerators that are completely different from the action required to select the same operation from the menu. Novice users do not naturally learn the expert way of activating a function. They have to make a conscious effort to learn a different way of activating operations. Further, the use of the keyboard may not be well suited to the user's task if most operations are performed with the mouse.

In a Control Menu the novice and expert uses of the menu are very similar. The only difference is that the novice pauses to see the menu and find the desired operation before moving the cursor. Novices thus rehearse the expert use of the menu every time they use the menu.

3.1.6 Menu is Transparent

A Control Menu (Figure 3.1) consists of (solid) black text drawn on top of red arrows that indicate where to move the cursor to select a given operation. The text is surrounded by a white border in order to increase its readability when it is drawn on a dark background. Control Menus consist only of these items so as to hide as little as possible of the user's workspace.

3.1.7 Activation Based on Distance

When using pull-down, pop-up, or marking menus, users indicate that they have chosen an operation and that this operation should now be executed by releasing the mouse button or lifting the pen. The activation is thus indicated by the end of gesture used to make the menu appear. When using a FlowMenu, users indicate that they have made their choice by moving the cursor back to a region in the

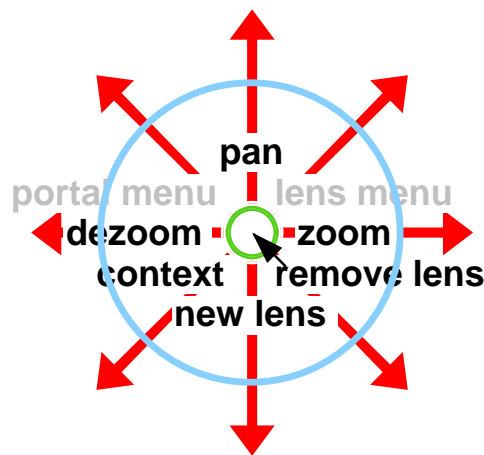


Figure 3.3: Activation based on distance

centre of the menu. Activation is thus determined by the position of the cursor relative to the menu. The gesture continues as the user has not had to release the mouse button and future mouse movements can be used to control the chosen operation or choose other operations

With a Control Menu activation depends on the distance that the cursor has moved from the point where the gesture started. As soon as this distance is passed the chosen operation starts. The light blue circle that passes through the “e” in “dezoom” in Figure 3.3 shows the activation distance in the Control Menu. As soon as the cursor crosses this (normally invisible) line the appropriate operation starts.

3.1.8 Discrimination Based on Angle

With pull-down and pop-up menus users indicate their choice of action by the distance they move the cursor before releasing the mouse button. With pie and marking menus it is the angle or shape of the movement made during the gesture that is used to determine which menu item is selected. The shape is analysed once the user releases the mouse button or stops moving the cursor (so as to see a menu). The distance moved is not important. FlowMenus use the sector traversed by the cursor before the return to the centre of the menu to determine which item is selected.

A Control Menu uses the angle of the mouse movement to determine which item is selected. As soon as the mouse is the activation distance from where the gesture started, the angle of the movement is analysed to determine the desired operation. The blue lines in Figure 3.4 (added to Figure 3.3) show the sectors associated with each item in the Control Menu. As soon as the cursor moves into

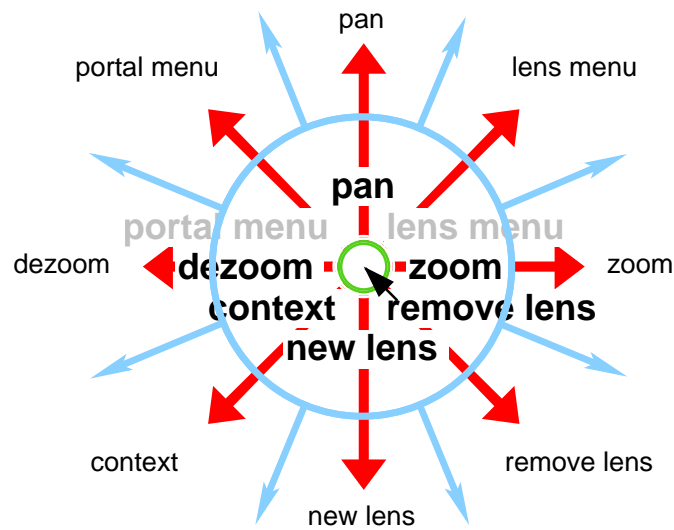


Figure 3.4: Discrimination based on angle

one of these regions, the operation whose name is indicated around outside of Figure 3.4 is started.

Textual strings are longer than they are wide. If they were to be drawn so that they fit completely within their region they would have to be drawn a long way from the centre of the menu. This would make the menu too big. As in marking menus, they can thus overlap the regions of adjacent menu items. Users do not seem to find this a problem as they tend to follow the large red arrow when they are learning the gesture to activate an item.

The combination of activation based on distance and discrimination based on angle means that choosing an operation with a Control Menu is as simple as pressing the mouse button and moving the cursor in the current direction.

3.1.9 Control of the Operation

The selection of an operation with a pull-down, pop-up, marking or pie menu requires users to release the mouse button. Any control of the chosen operation requires the use of another interactor. This interactor is often a dialog box and its use is modal. The interface is blocked until the dialog box is dismissed. Modal interfaces are generally accepted to be error prone and confusing for users (Raskin, 2000). The use of two interactors to perform a single operation is also undesirable as users have to focus their attention on the menu and then on the other interactor before having to return to what they were working on.

FlowMenus allow the chosen operation to be controlled in the same gesture used to choose the operation. This control can be either via fixed choices from a

menu or via movements in a circle around the centre of the menu. This quasimode can be left by returning to the centre of the menu. The control in one dimension provided by a FlowMenu only uses one of the degrees of freedom of the mouse but can be used on very small displays.

Control Menus use both of the mouse's degrees of freedom to control the chosen operation. The cursor can be moved anywhere on the screen as there is no region that causes the control quasimode to stop. This control quasimode continues until the mouse button is released. As interaction that with a FlowMenu requires a circular action can be provided by a Control Menu with a linear action. A linear action is easier to perform than tracing a circle but requires more space.

Control Menus thus provide more control possibilities but it is not possible to leave the quasimode other than by ending the gesture. This means that it is not possible to choose and execute a number of different operations in a single gesture.

3.1.10 Comments

Control Menus have some features in common with currently used menus such as pull-down, pop-up, pie and marking menus. They also share some of the characteristics of FlowMenus which were developed after Control Menus. The control possibilities offered by Control Menus are however quite different and are discussed in the following sections.

3.2 Controlling Operations

Control Menus are particularly useful for controlling operations with one or two numeric parameters with a large number of possible values. With this type of parameter the control part of the use of a control menu is simple.

3.2.1 Operations With One Parameter

If the operation has only one parameter then as soon as the operation has been selected mouse movements in one direction increase the value of the parameter and mouse movements in the other direction decrease the value of the same parameter. Any two opposing movements can be used, for example horizontal or vertical.

Example: Zooming

Figure 3.5 shows the gesture used to control the level of zoom (either of an entire application or of just the object at the cursor position at the beginning of the ges-

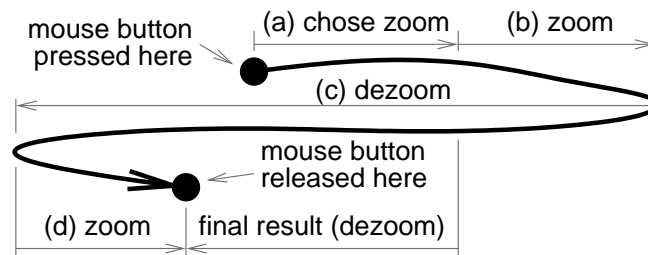


Figure 3.5: Zooming with a Control Menu

ture). The zoom operation is on the right hand side of the Control Menu used in this example; horizontal mouse movements are to be used to control the operation. Movements to the right increase the zoom and movements to the left decrease the zoom.

There are two main parts to this gesture. The first, Figure 3.5a, selects the zoom operation. As soon as the cursor arrives at the end of this part of the gesture the zoom operation starts and the menu vanishes (if it was visible). All subsequent mouse movements control the operation. The movement Figure 3.5b increases the zoom level and the movement Figure 3.5c then decreases the zoom level below that at the beginning of the operation. The movement Figure 3.5d increases the zoom level somewhat but the final result, when the mouse button is released, is a dezoom.

Replaces Selection, Menu, Slider and Button

This use of a Control Menu shows how a single use of a Control Menu can incorporate the choice of a target object, the choice of an operation, a slider to control the operation and the button used to indicate that the desired value has been found. This reduction of four interactors into a Control Menu means that users need only one interactor for even quite complicated tasks. Users avoid the focus of attention changes required by multiple interactors.

Undo is Possible

Operations that have only one parameter to be controlled during the gesture use only one of the two degrees of freedom of the mouse. The other degree of freedom can be used to provide other capabilities. One possibility is undo. If horizontal movements are being used to control the scale of the interface then a vertical movement could undo changes made to the scale during the current use of the Control Menu. The undo could of course be undone by returning the cursor to the original horizontal position. Undoing and redoing the scale change would allow

users to compare the original scale and the new scale before deciding which to keep.

Zoom to Dezoom

Figure 3.5 shows that a menu item to change a parameter in one direction, zoom, can also be used to change the parameter in the other direction, dezoom. If space is at a premium in a Control Menu and there is not enough space to include zoom and dezoom entries in the menu, just one of these entries is sufficient. The (slight) disadvantage is that, as in this example, a dezoom has to start with a small, and rapidly corrected, zoom.

Some users found the use of only one menu entry for zooming and dezooming to be confusing. They quickly learnt that zooming meant moving to the right and dezooming to the left. They forgot that moving to the left dezoomed only once the zoom operation has been selected, i.e. after a move to the right. The Control Menu was modified so as to have complementary zoom and dezoom menu items. Users can still dezoom with the zoom command but they can also dezoom directly.

3.2.2 Operations With Two Parameters

Most pointer devices used today, such as mice, trackballs, graphics tablets, and pens, have two degrees of freedom. These devices can supply two independent values to the computer systems. Some devices such as pens on small touch sensitive screens may have two degrees of freedom but only over a limited distance. In this case it may be necessary to reduce the two degrees of freedom to one by requiring users to make circular movements in the limited space available.

Example: Panning

When there are two degrees of freedom, these can be used by the control menu to control two parameters simultaneously. The simplest example is controlling the x and y coordinates of the position of an object or a viewport. In this case a Control Menu is used to pan (or scroll) in two dimensions thus replacing two scroll bars. The pan operation is selected by pressing the button and moving the mouse up (Figure 3.1 shows the pan entry at the top of the menu). During the operation, i.e. until the mouse button is released, the view follows the cursor. The solid line in Figure 3.6 shows the mouse movements during the choice and execution of the pan operation. The first movement, which can be performed immediately or after the pause to see the menu, chooses the pan action. Once the action has been chosen, the view follows the mouse. The second movement is towards the north and moves the view towards the north. The subsequent mouse movements

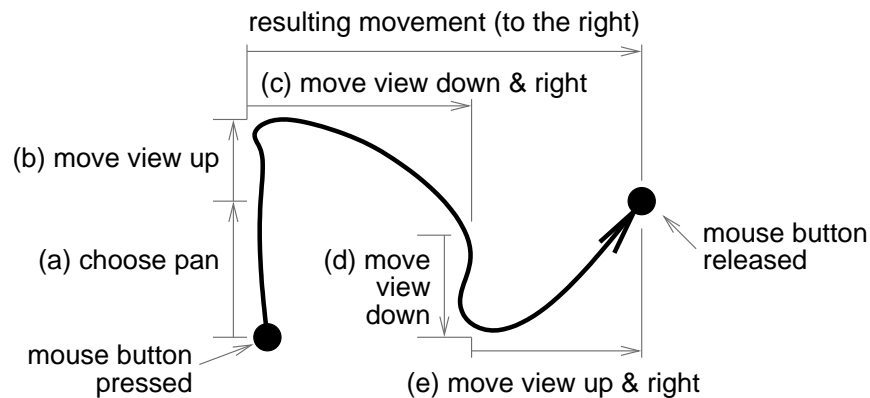


Figure 3.6: Panning with a Control Menu

continue to move the view with the cursor until the mouse button is released. The net result of the action is a movement of the view towards the right (or east).

Go Up to Go Down

The position of the pan operation on the menu requires users to move the cursor towards the top of the screen to start the pan operation. Users thus move the image up (a little) even when they want to move it down. This situation is similar to that described in section 3.2.1, where users have to zoom (a little) to dezoom, but it appears to be less of a problem for users.

No Undo

The pan cannot be cancelled during the operation because all the possible movements of the mouse already have a meaning. The possibility to cancel the operation could be provided via the use of a different input device such as the keyboard or another mouse button (if there is one).

Integral Parameters Work Best

Control Menus work best when controlling two parameters that are integral. Integral parameters (Jacob and Sibert, 1992) are those whose attributes combine to form a single composite attribute in the user's mind. The x and y coordinates of an object are integral parameters because users combine them into the position of the object. A diagonal mouse movement has a simple meaning in this situation: move the object diagonally. On the other hand, the size and colour of an object are not integral parameters. If a Control Menu is used to control two such param-

eters simultaneously, a diagonal movement of the mouse does not have a simple meaning.

3.2.3 Comments

Control Menus have a number of properties that aid users in controlling computer systems.

Parameter Value Visible

The modification of the parameter starts when the operation is selected. If the value of the parameter is associated with the position of the cursor then the value is “jumped” to correspond to the position of the cursor when the operation starts. This is the case if a Control Menu is being used to move a particular object. The object should remain under the cursor.

If no visible object represents the parameter being modified then it is not necessary to “jump” the value of the parameter to take into account the distance moved (Figure 3.5a) during the selection of the operation. This is the case if the Control Menu is being used to change a global zoom factor or to pan the entire view.

Fine Control

The maximum change of the parameter in any direction is limited by the range of movements of the mouse. Mouse movements can normally be tracked even when the cursor has left the application’s window. The range of mouse movements is thus limited by the space available on the user’s desk for mousing. The parameter value is updated with each movement of the mouse (no matter how small). This allows fine control of the value of the parameter limited only by the resolution of the mouse and the space available.

Immediate Visual Feedback

In most uses of a Control Menu the representation of the interface or the object being changed is updated after each and every mouse movement. This gives the immediate visual feedback that allows users to continue adjusting the parameter until they see exactly the desired result. At that point they just have to release the mouse button to keep the desired value.

Where updating the user’s view leads to large and possibly disorientating changes it may be better not to update the view after each mouse movement. This

is the case where a Control Menu is being used to select and delete text. Interactively deleting each character as the user moves the cursor will lead to continuous reformatting of the document that may be distracting. It may be better, during the gesture, merely to indicate what text is to be deleted and then to delete it at the end of the gesture.

Temporary Parameters

If the parameter being controlled is temporary (it exists only during the gesture) and has a limited range of values, and this use of the menu is not contextual, then the position of the cursor at the beginning of the gesture can give the initial value of the parameter. In this case, moving the cursor to right hand side of the application's window increases the parameter to its maximum value, and moving it to the left hand side will set it to its minimum value. This possibility is described in more detail in the section that discusses the use of Control Menus in Zoomable User Interfaces (subsection 5.2.8).

The control of temporary parameters in this way differs from that normally used in Control Menus. In normal use moving the cursor increases or decreases the parameter. The change in the parameter caused by a given movement of the input device is determined by the tradeoff between precision and rapidity. The maximum change in a single use of a Control Menu is limited by the range of movements provided by the input device. When controlling temporary parameters as described in this section, the maximum value is obtained by moving to one side of the screen and the minimum to the other. Each cursor position within the application's window corresponds to a fixed parameter value.

Cursor Indicates Choice

The operation that has been chosen and is active is normally obvious because of the immediate feedback to any mouse movement. As an extra aid to reassure users that the current operation has been chosen, the cursor changes as soon as an operation starts. Each operation has its own cursor. Users can just look at the cursor to check that the correct operation has been selected.

On Screen Guides

The techniques described so far in this chapter are designed for interactions where the effect on the interface of moving the cursor during the control phase of the gesture is obvious or is easily discovered with the help of the visual feedback. If this is not the case, it is necessary to help users understand the effect of mouse

movements by drawing a transparent aid on the screen. This aid indicates in which direction users should move the cursor to obtain a desired result.

Such an aid is even more important if it is not just what effect is to be produced by moving the cursor that is non obvious, but also where in a given direction it is necessary to position the cursor to obtain a given result. This will be the case if a Control Menu is being used to select between values that have no generally recognised order. When selecting between numeric values, such as a font size, it is generally expected that moving to the right chooses a higher value. If the movement is vertical, it is perhaps also obvious that higher values are obtained by moving the cursor up. When a Control Menu is being used to select between different fonts, for example, and the current font is Helvetica, it is not at all obvious which way users should move the cursor to select the font Palatino. A guide indicating the position of the different fonts along the chosen axis will have to be displayed during the use of a Control Menu to select a font. In this situation, the Control Menu incorporates the functions of a set of radio buttons rather than a slider.

3.3 Sub-menus

A Control Menu can contain sub-menus. A sub-menu is opened the same way that an operation is chosen: by moving in the direction of the sub-menu's entry. As soon as the cursor has moved the activation distance the sub-menu is opened. An open sub-menu is shown in Figure 3.7. The sub-menu is linked to its entry in the main menu by a solid red line. The corresponding entry in the main menu is also shown in a different colour (green in this example) so as to remind the user which menu is open. If the sub-menu was opened in error, it can be closed by moving the cursor back to the centre of the main menu. The entire action of choosing an operation can be aborted either by closing the sub-menu and releasing the button with the cursor in the middle of the main menu or by directly releasing the button with the cursor in the middle of the sub-menu. The circle in the centre of the sub-menu in Figure 3.7b has turned green to indicate that no operation is currently selected.

Once a sub-menu is open it is used in the same way as a top level menu with the difference that moving the cursor into the small circle in the centre of the main menu will cause the sub-menu to close.

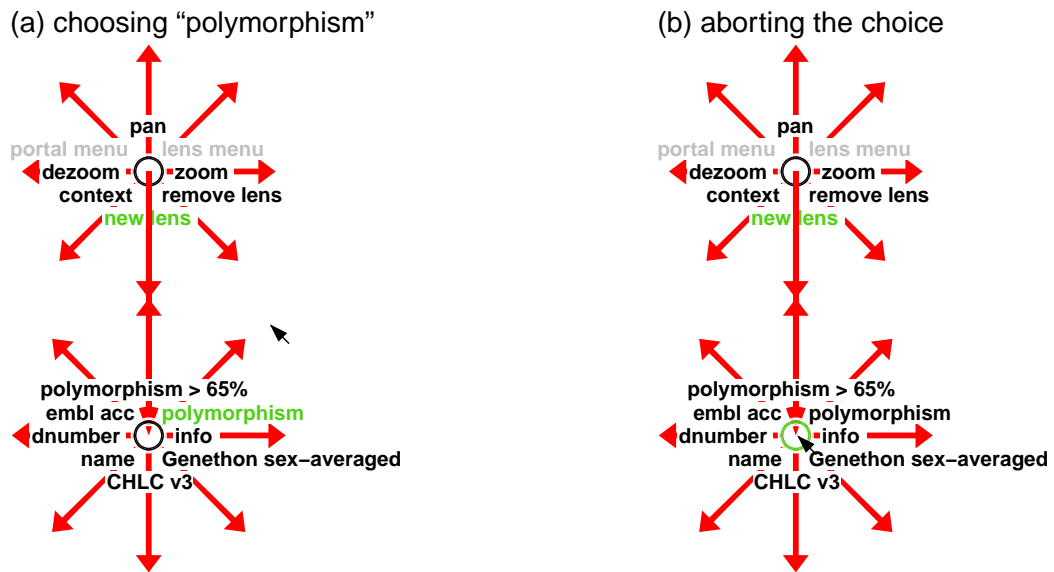


Figure 3.7: Lens sub-menu in our Zoomable User Interface

3.4 Buttons

A Control Menu can contain simple commands that do not have any parameters. Commands that can be cancelled are executed as soon as the cursor has been moved the activation distance from where the mouse button was pressed. As the user still has the mouse button pressed, a movement in the other direction undoes the operation. The undo can be reversed by moving the cursor back again. The user releases the button when the desired result has been obtained. A Control Menu could be used in this way, for example, in a text editor for the operation of deleting text. It could not be used for the save operation because saving cannot be reversed.

Commands that cannot be cancelled are provided an extra level of protection. When the user moves into the sector associated with such a command the command does not start immediately. The menu is drawn if it is not already visible and the name of the command changes colour (green in our implementation) to indicate that it will be executed if the user releases the mouse button at that moment. User can thus change their minds by moving the cursor back to the centre of the menu. Figure 3.7 shows a situation where the interface will create a “polymorphism” lens if the mouse button is released. The user can decide to not execute any operation by moving the cursor as shown in Figure 3.7 and releasing the mouse button.

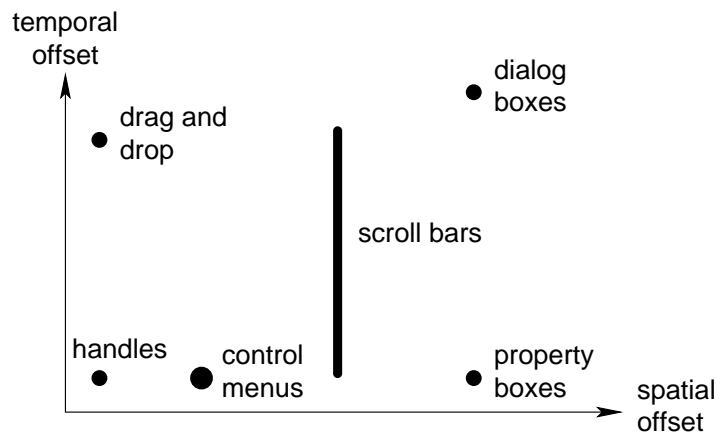


Figure 3.8: Degree of indirection

3.5 More Than a Menu

A Control Menu is more than a simple menu because it includes the possibility of controlling the chosen operation. It thus incorporates one or two sliders (or sets of radio buttons) that would normally be in a dialog box. Control Menus can thus be analysed by a general interaction model and with regard to Fitts' Law.

3.5.1 Interaction Model

Control Menus can be analysed by the interaction model proposed by Beaudouin-Lafon (2000). This model was described in section 2.3

Degree of Indirection

Figure 3.8 shows how a Control Menu fits into a diagram that shows the degree of interaction of various interactors (Beaudouin-Lafon, 2000). Control Menus change the underlying object immediately with a visual feedback that shows changes as they occur. Their temporal offset is thus low. The spatial offset is also initially low as the menu is contextual and must be used over the target object. The spatial offset can however increase from the initially low value if users use it to modify a parameter of an object that does not cause the object to move or grow at the same rate at which the cursor moves. An example where the spatial offset remains low is when the Control Menu is moving an object: the object follows the cursor. If a Control Menu is being used to change the colour of an object, the distance between the object and the cursor will become non-zero since the object will not grow as users move the cursor to chose the new colour.

Degree of Integration

A Control Menu can use all the degrees of freedom of an input device. This thesis has only studied the use of Control Menus with input devices that have two degrees of freedom, there is however no reason to suspect that they would not be usable with input devices with multiple degrees of freedom, especially if the parameters being controlled are integral.

It may also be possible to create a Control Menu that has more degrees of freedom than the input device. This could be done by deciding that diagonal movements of the cursor, with a two dimensional input device, should not affect the vertical and horizontal parameters but should rather change the values of two independent diagonal parameters. More work needs to be done in this area. This could also be the solution to the problem of non-integral parameters.

Degree of Compatibility

The degree of compatibility of a given Control Menu depends on the degree of compatibility of the mouse movements in the control phase of the gesture and the result of changing the corresponding parameters. If a Control Menu is being used to move an object or to pan a view, the degree of compatibility is high since the object or view follows the cursor. If a Control Menu is being used to zoom a view, the degree of compatibility is much lower as there is no clear correspondence between the movements of the cursor and the movements of the object in the view.

3.5.2 Fitts' Law

Fitts' Law (Raskin, 2000) states that the further a target is from the user's current cursor position or the smaller the target, then the longer it will take the user to move to the target.

With a Control Menu the distance from the cursor to the menu is zero; the menu pops up under the current cursor position. The distance from the centre of the Control Menu to the regions that choose operations is the activation distance given in subsection 3.1.7. This distance does not depend on which item in the menu is to be chosen. The fact that all menu items are equally close to the cursor means that it is not necessary to decide where items should be placed as a function of their frequency of use. The position of items in a Control Menu can be chosen by taking into account the semantics of the operation, the meaning that users might give to the movement required to choose an item in a particular position in the Control Menu, and the movements then used to control the operation. An operation that moves objects horizontally should probably be on the left or right

of the Control Menu. There is some debate on whether a zoom operation should use horizontal or vertical mouse movements but its position in the menu should be same as the movement in the control phase.

The targets in a Control Menu have infinite size because users can move as far as they wish in the correct direction. They do however have to be able to move in the correct direction. The accuracy with which they need to move the cursor depends on the number of items in the menu.

3.6 Applications

We developed Control Menus to improve the control of Zoomable User Interfaces and to investigate the synergy between control and visualization in these interfaces. Control Menus are however general interactors and we have used them to control the navigation in virtual worlds. We discuss in this section how they could be used in more standard programs such as text editors and also how they could be adapted to systems with limited screen space.

3.6.1 Zoomable User Interfaces

Zoomable User Interfaces are complex programs with complex user interfaces that are normally controlled using the mouse, buttons and standard menus. When navigating in a ZUI, users zoom, dezoom, scroll, create Magic Lenses, move and resize Magic Lenses, move and scroll portals, etc. Some of these actions are executed very frequently. A user zooms until the desired scale has been obtained and scrolls until the object looked for has been found. The graphical objects used to present the information space change frequently; a zoom or dezoom can completely change the visible objects. Making these objects active and using them to control the ZUI can lead to unwanted operations being chosen as these objects change and move frequently. Our Control Menu is well adapted to be used in Zoomable User Interfaces.

We describe the use of a Control Menu to control the virtual worlds constructed with our Zoomable User Interface development tool, Zomit, in chapter 5. We also describe how the fluid integrated selection and control possibilities of Control Menus allow new types of interaction to be envisaged.

3.6.2 Interaction in a Virtual World

A modified Control Menu is used to navigate in the *vreng* virtual world (<http://www.infres.enst.fr/net/vreng>) developed at the École Nationale Supérieure des Télécommunications (Figure 3.9). The Control Menu can be used to teleport the

-
- (a) [His tender heir might bear his memory
But thou contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
- (b) [His tender heir might bear his memory:
But thou contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
- (c) [His tender heir might bear his memory:
But thou contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,

Figure 3.10: Selecting text with a Control Menu

two steps. The text to be changed is selected via the menu and then the “change font size” operation is selected with a second use of the menu. This operation has one parameter: the change in the point size of the text. Moving the cursor upwards increases the size; moving the cursor downwards decreases the size. The user releases the button when the correct size has been found. As the user moves the cursor, the selected text changes to the new point size. To avoid distraction, and perhaps computationally expensive layout changes, the user’s document is not reformatted to take into account the new point size during the gesture. Rather, the selected text is shown in the new size, in the space occupied originally by the text. The document’s layout is updated once the user ends the gesture by releasing the mouse button. This process would be different in a program for preparing presentations. Here there is much less text to be reformatted after a font size change and the visual aspect of the page is more important. The change in font size would thus be performed during the operation so that the user can find the desired visual effect.

Operation such as “cut” or “underline” which, like selecting text, do not require any parameters, can be provided in the same way as a font size change (two menu accesses) or via a specialised menu command which allows the user to select text and then removes or underlines the text at the end of the operation. An underline command would underline text as it is selected. To avoid continual formatting when deleting, the deleted text would vanish as it is selected but the document is reformatted only at the end of the operation. Simple commands can thus be performed with a minimum of menu use.

A Control Menu would be most useful in a presentation editor. Here the choice of visual effects is larger than in text editors where they are often imposed by pre-defined styles. In a presentation editor the immediate feedback provided by Control Menus allows users to see and adjust in a single gesture the visual effect that they are creating. When used to select colours, fonts, font styles and other visual effects where ordering of the possible values of the parameter is not obvious, a visual guide would be required (subsection 3.2.3).

3.6.4 Interfaces With Limited Screen Space

It is possible to use a Control Menu on a device that has very limited range of movement for the pen, such as a mobile phone screen. In this case the selection part of the Control Menu is used normally but during the control phase the two degrees of freedom of the pan are used to give one movement dimension. This is done by moving the pen in circles around the centre of the Control Menu. Clockwise movements increase the parameter value and anticlockwise movements decrease the value.

3.7 Discussion

Control Menus fill a gap in the design space of menus. They provide rapid selection of operations in a way similar to pie and marking menus but also allow the control of the chosen operation in the same gesture. Selection and control are thus combined, leading to a more fluid interaction between humans and the computer programs that they use.

Part II

Visualization

Chapter 4

Visualization Research

Databases of many different types are already very large and have been growing in size at an exponential rate for years. The databases contain information spaces that users need to understand, interrogate and control. Visualization systems have been developed to aid users in these tasks. One of the principal problems is that these spaces are so large that traditional techniques are no longer sufficient for presenting the entire space to users on computer screens that have increased in size only slightly.

The small part of the information space that does fit on the screen is such a tiny fraction of the entire space that users find it difficult to maintain an understanding of the position of their view in the global information space. The visualization problem is to provide sufficient context for users' small view into a huge databases.

Bartram et al. (1995) state that the solutions to alleviate the visualization problem typically fall into one of three general classes: traditional pan and zoom, multiple window (or map view), and distorted view.

The traditional pan and zoom technique has a single but not necessarily constant scale for the entire space. Users pan and zoom in the space looking for the piece or pieces of information that interest them. The problem with this technique is that if the information space is large users will soon get lost because they do not have any context to help them understand how the information that they are currently looking at fits into the complete information space.

The multiple window solution provides an overview window in addition to one or more detailed views. The different detailed views can show different regions of the information space or the same region in a different format. This solution consumes additional space for the overview and requires users to mentally integrate the different windows into a complete picture of their position in the information space.

Distorted view methods (so-called “fisheye” views) deform the information space in order to show certain regions (the focal area) in detail while keeping the larger context visible in the same window. We present some distorted-view techniques and explain why their limitations mean that they are not ideal in some visualization scenarios.

The visualization of hierarchies or trees is a subject of extensive study in the literature. We present a description of some visualization systems for trees as an example of the visualization of a particular type of data. These systems will illustrate the taxonomies presented at the end of this chapter.

Another new technique is the use of transparency. This technique introduces a new dimension into the user interface: depth. The use of depth allows multiple views to be shown at the same time and in the same screen space. Users must be able to distinguish the different views.

Multiscale Interfaces, also known as Zoomable User Interfaces (ZUIs), are derived from the traditional pan and zoom. These interfaces allow users to control the scale of the presentation of the information space. ZUIs use semantic zooming: when displayed objects receive more screen space as users zoom, the representation of the objects changes so as to show the objects in more detail. This technique differs from the multiple window solution in that it is “immersive”. When users zoom the entire view zooms and they plunge into the information space.

The traditional pan/zoom and multiple window techniques are well understood and used in current commercial visualization systems. They do however have their limitations and so we chose to concentrate on the new visualization methods and discuss how they can help to solve the visualization problem. We thus present the other types of systems mentioned in the introduction followed by summaries of several known taxonomies. We then present two new taxonomies. The first examines how information can be presented: with a time, depth or space multiplexed presentation. The second describes the different types of deformations that can be used to multiplex the user’s view.

4.1 Distorted Views

The techniques described in this section are based on the idea that users, for a given task or at a given instant, have a focus of attention (the object or objects that they are currently working on) and that the other objects, non-focal, are less interesting. These “less interesting” non-focal objects are still important as they allow the users to position the focus in the information space. It is also assumed that the further the non-focal objects are from the objects of the focus of attention, then the less interesting they are. A representation of these non-focal objects is however required as a context for the focal objects. These techniques thus relate the screen

area given to non-focal information to the distance from information to the focus. The further information is from the focus the less interesting it is assumed to be and thus the smaller it is shown. These methods deform the information space by eliminating information or by changing the size and position of the representation of information.

In these systems the deformation is not constant over the information space. The further a piece of information is from the focus the more it is deformed.

It is possible to generalise these views to include the possibility that users have several non-adjacent centres of attention. This leads to multiple foci and more complicated distance and interest functions.

4.1.1 Fisheye Views

Fisheye views (Furnas, 1986) are one way of integrating the context and focus into a single view.

A Degree Of Interest (DOI) function assigns, to each point in the structure to be visualized, a number indicating how interested the user is in seeing that point, given the current task. A basic strategy to display information on a screen of size n , is to display the n most interesting points, as indicated by the DOI function. Furnas (1986)'s generalised fisheye views decompose the DOI function into two components: $d_f(x \vdash y) = i_a(x) - D(x, y)$, where $i_a(x)$ is the *a priori* interest in a point x , $D(x, y)$ is the distance between x and y , and $d_f(x \vdash y)$ is the user's degree of interest in x given that the current focus is y . The interest decreases with distance: the greater the distance of the information from the focus the more interesting the information must be for it to be shown. This formulation allows fisheye views to be defined in any sort of structure where $i_a(x)$ and $d(x, y)$ can be defined.

Figure 4.1 shows a fisheye view of a C program (Kernighan and Ritchie, 1989). The line numbers are shown on the left with the user's current line indicated by " \Rightarrow ". The " \dots " indicates where lines have been removed. All of the code in the current case is shown, as are all the control structures that enclose the current line of code and all the declarations of variables that are visible. Other control structures (lines 47 and 57) in the immediately surrounding block are also shown.

Noik (1993) builds on Furnas (1986)'s work and discusses how to browse a typical hyperdocument. The Canada-U.S. Free Trade Agreement is a medium-sized hyperdocument. It has 1680 nodes and 3852 links of three kinds: containment (e.g. from a part to a contained chapter); sequencing (e.g. the original linear ordering); and, semantic (e.g. cross references).

4.1.2 Perspective Wall

Mackinlay et al. (1991) present the Perspective Wall as a way of visualizing linear

```

1  #define DIG 40
... 2  #include <stdio.h>
4  main()
5  {
6      int c, i, x[DIG/4], t[DIG/4], k = DIG/4, noprint = 0;
... 8      while((c=getchar()) != EOF){
9          if (c >= '0' && c <= '9'){
... 16         } else {
17             switch(c) {
... 18                 case '+':
... 27                 case '-':
... 38                 case 'e':
⇒ 39                     for(i=0;i<k;i++) t[i] = x[i];
40                     break;
... 41                 case 'q':
... 43                 default:
46             }
... 47             if(!noprint){
57             }
58         }
59         noprint = 0;
60     }
61 }

```

Figure 4.1: Fisheye view of a C program (adapted from Furnas, 1986)

information that involves spanning properties such as time. Perspective Walls convert an arbitrary 2D layout into a 3D visualization by folding the left and right sides of the 2D layout away from the user (Figure 4.2). (The 2D layout is assumed not to be taller than the screen. This method cannot handle layouts large in both dimensions.) The central part of the 2D layout remains undistorted while the sides are represented in perspective, the objects on the sides are smaller the further they are away from the user. Users can transfer the focus of attention by moving the folds. They can transfer the focus of attention to the objects on left of the 2D layout by moving the folds to the left. More of the objects on the left are thus shown on the undistorted centre region and some of the objects previously on the centre area are moved off onto the right side wall. This transition between views is smooth; the user sees the objects move from the centre region to one side and new objects move from the other side into the centre. The transition is even easier to follow if the transition is animated. The ratio of detail and context can be adjusted by the user by increasing or decreasing the size of the centre (focal) region.

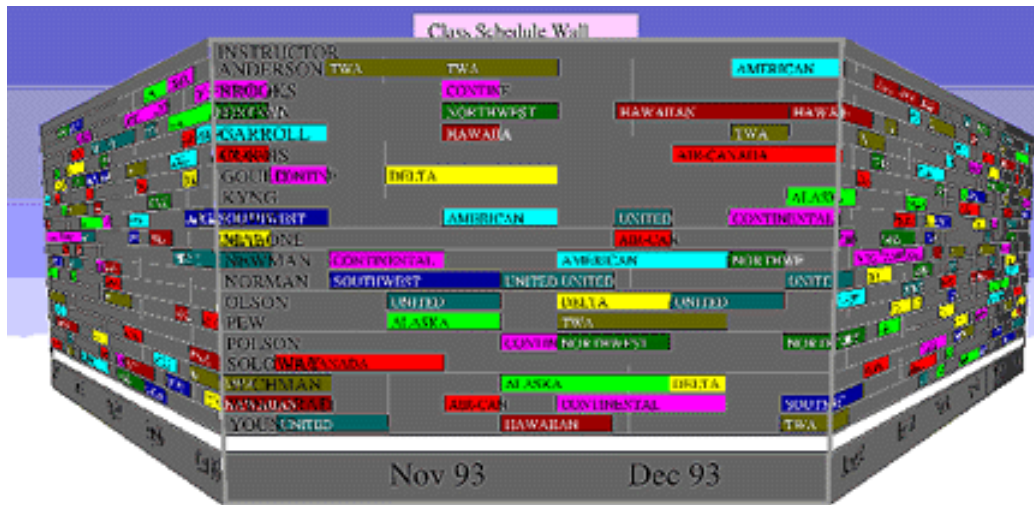


Figure 4.2: Perspective Wall (from www.parc.xerox.com)

The Perspective Wall can be used to visualize data that is linear in form, spanning for example a period of time. An example could be data from different departments (the rows) over a large number of months (the columns). At any moment, the user has a number of months easily readable on the centre area. The data from the other months are shown smaller; the further away from the centre (the focus of attention) they are, the smaller they are shown.

One obvious problem with this visualization method is that valuable screen real-estate is wasted in the corners. The perspective view implies that the heights of the side walls diminish the further away they are from the user. Mackinlay et al. (1991) precede their presentation by citing Resnikoff (1989) who says that the retina of the human eye has only a limited centre region that perceives details.

The user is limited to one region of focus. It is not possible to compare two widely separated regions.

This paper lists the two basic strategies for solving the visualization problem. The first is called a *space strategy*, where all the information is shown at all times, thus a method has to be found to make the information fit on the small screen while keeping at least the focus of attention readable. The Perspective Wall is a space strategy. The other is the *time strategy* where different views, presented at different times, allow access to all the information. Here the systems needs to help prevent users getting lost as they flick from view to view.

A possible extension to this system would handle layouts that are high as well as large. This would require perspective views in the horizontal and vertical dimensions. The resulting deformation would look somewhat like that proposed by the Document Lens.

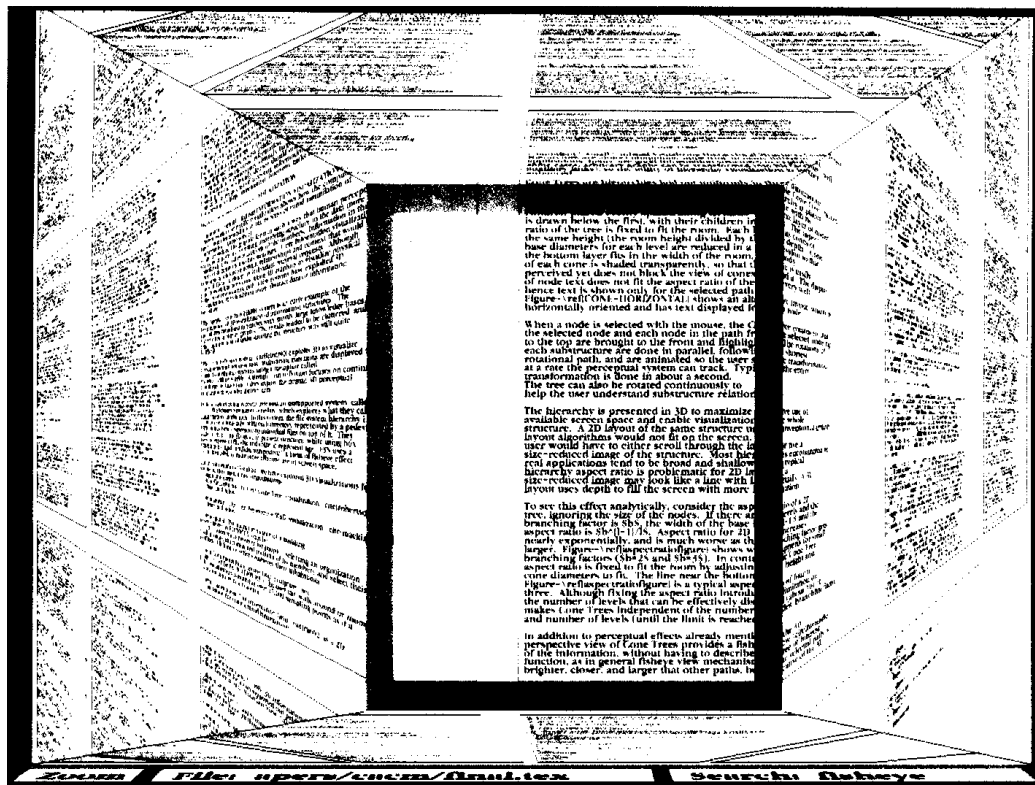


Figure 4.3: Document Lens (adapted from Robertson and Mackinlay, 1993)

4.1.3 Document Lens

The Document Lens (Robertson and Mackinlay, 1993) allows the user to focus on a part of a document while keeping the surrounding pages (the context) visible and can be used to view a very large document laid out in a rectangular array. (A rectangular array is a suitable form to view a document, as opposed to the linear array proposed by Mackinlay et al. (1991), because it fills the screen when used to view a document.) With the entire document visible on the screen the text will not be readable. The system converts the flat document into a pyramid with its top cut off. The flat area at the top of the pyramid is the focus and contains easily readable text. The text on the sides of the pyramid closest to the top is also readable, giving extra context, and the entire document always remains visible. Users use the mouse to move the lens (or top of the pyramid) in the x - y plane and the keyboard to move the lens backwards and forwards (to make the lens bigger or smaller). The problem is that not much of the non-focus text is readable, especially when the lens is close to the user (and thus large) and the context is lost.

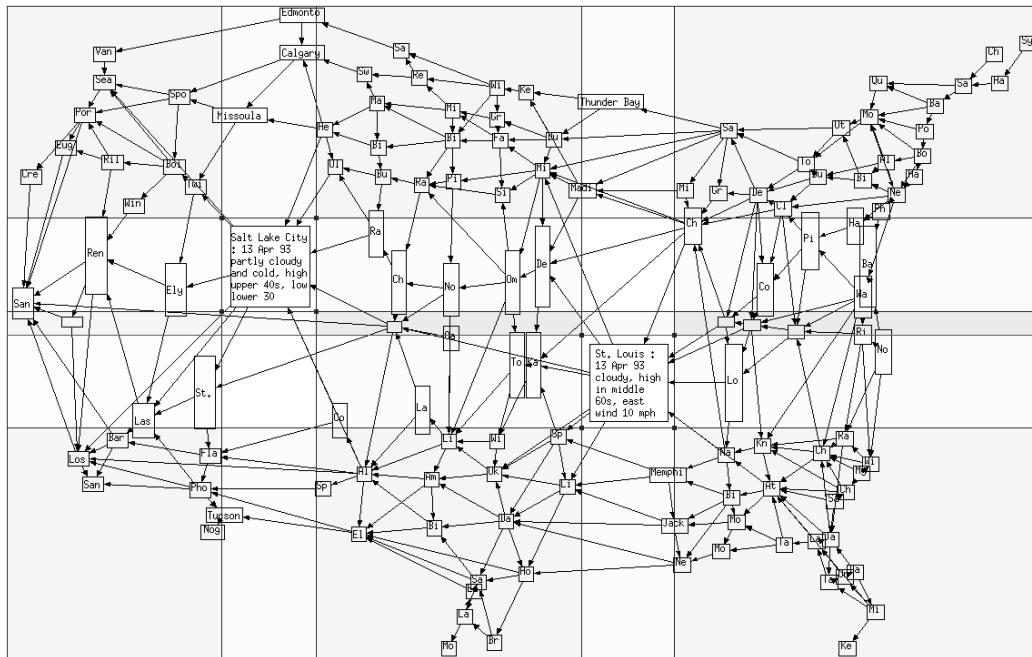


Figure 4.4: Rubber Sheet with orthogonal stretching (adapted from Sarkar et al., 1993)

Carpendale et al. (1997b) separate and analyse the different components used in visualization approaches such as the Document Lens. The two principal components are magnifying selected areas (the focus) while adjusting the relative positioning of adjacent sections in order to avoid congestion and overlap.

4.1.4 Rubber Sheets

Sarkar et al. (1993) describe a system where the screen is seen as a stretchable (rubber) plane. The user can enlarge (stretch) any part of this sheet in one of two ways.

The first method, called *orthogonal stretching* (Figure 4.4), is to surround the area of interest with two vertical lines (that cover the full height of the screen) and two horizontal lines (that cover the full width of the screen). The area enclosed by these four lines is enlarged in both dimensions, while the areas to the left, right, top, and bottom of the central area within the lines are enlarged in only one dimension. All areas not within these lines are reduced in size by the same factor. It is possible to add additional lines to create additional areas of focus. The main disadvantage of this method is that the regions close to a focus area are just as small as remote regions and that there is a discontinuity of scale between a focus

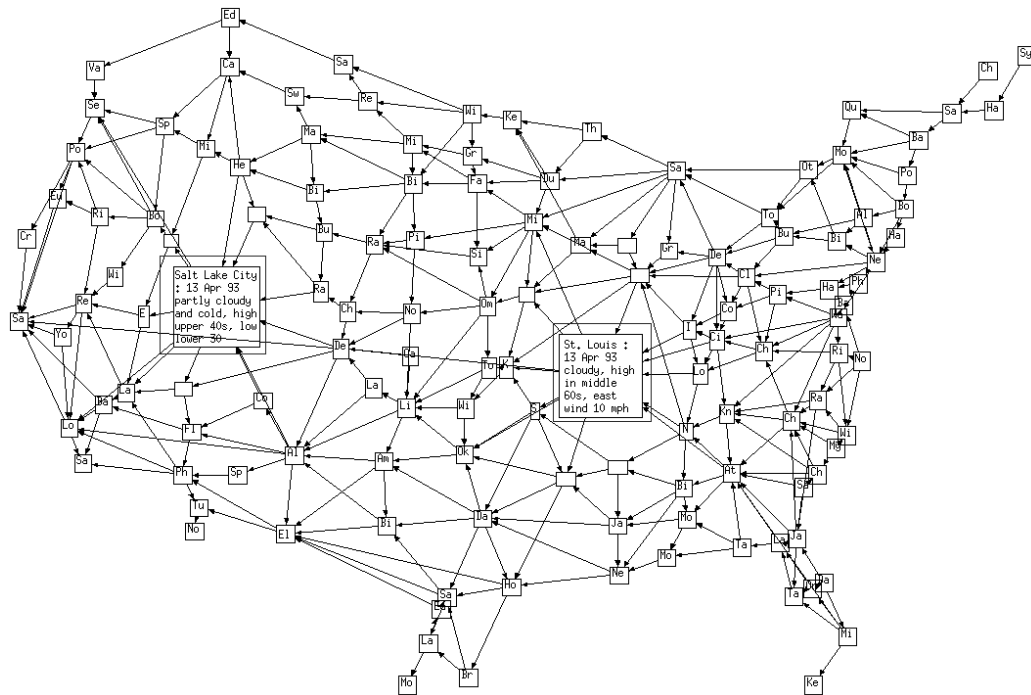


Figure 4.5: Rubber Sheet with polygonal stretching (adapted from Sarkar et al., 1993)

area and adjoining regions. A second disadvantage is that the row and column based system for selecting a focus area means that the regions in the same row and column are enlarged (in one dimension) as well. Orthogonal stretching does have the advantage that the orthogonal ordering of points is maintained.

The paper presents a second method called *polygonal stretching* (Figure 4.5). The user chooses an arbitrary area to enlarge. Those regions nearby are slightly enlarged, those further away reduced in size (to make enough space) and far distant regions are not affected. This non-orthogonal distortion does not maintain the relative order of objects. An object that in the undistorted view is below and to the right of another object, can, after the distortion, be found to the left of the other object. Those parts of the view, not close but not adjacent to the focus, that are reduced in size might not be readable. This is a problem because these might be just those objects the user needs to see.

Editing in the focus area is possible with orthogonal stretching and, if the focus is circular or rectangular, with polygonal stretching.

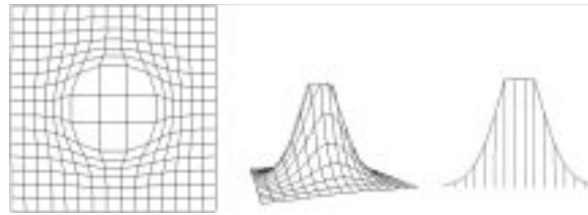


Figure 4.6: Single focus 3D pliable surface with flattened top (adapted from Carpendale et al., 1995)

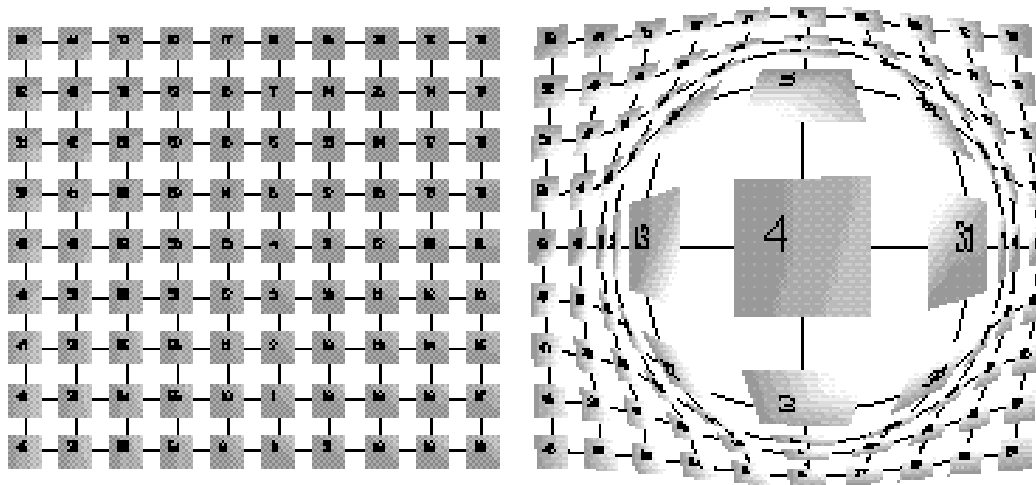


Figure 4.7: Magnification and distortion with 3D pliable surfaces (adapted from Carpendale et al., 1995)

4.1.5 3D Pliable Surfaces

Carpendale et al. (1995) discuss the use of three-dimensional pliable surfaces in viewing graphs and maps. They use Gaussian curves to transform the two-dimensional flat surface containing the graph or map into a three-dimensional curved surface. This surface is then viewed from above (Figure 4.6). The user can modify the distortion applied to the flat surface by, for example, pulling on a section to magnify it (thus adding a Gaussian curve to the distortion). Figure 4.7 shows a grid and a distorted view of the same grid showing the distortion and displacement caused by the magnification of the centre region. The overall appearance of a multi-focal view can be seen as a curved “landscape” (Figure 4.8). The system uses a regular grid (drawn on the two-dimensional surface and transformed with it) to indicate the three dimensional form of the surface. The authors also considered using shading or a three-dimensional perspective.

Carpendale et al. (1997a) propose a summary of the different distortion pat-

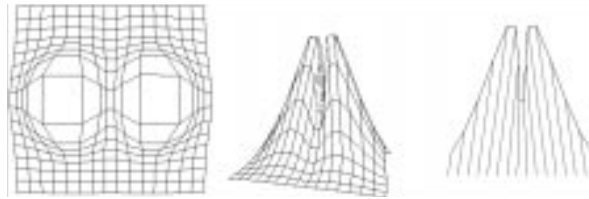


Figure 4.8: Double focus 3D pliable surface (adapted from Carpendale et al., 1995)

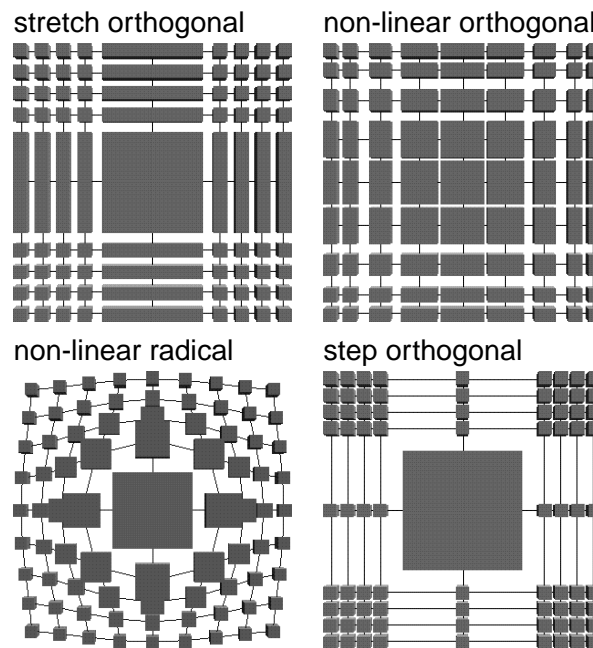


Figure 4.9: Distortion patterns (adapted from Carpendale et al., 1997a)

terns used in 2D visualization systems such as Rubber Sheets (subsection 4.1.4). They show how the non-focal parts of an information space can be distorted when space needs to be found for a magnified focal area. Figure 4.9 shows four different transformations in 2D. Figure 4.10 shows other, less frequently used, 2D distortion patterns.

4.1.6 Table Lens

The Table Lens (Rao and Card, 1994, 1995) is a method of visualizing and making sense of large tables such as those found in a spreadsheet. It is an adaption of rubber sheets with orthogonal stretching (subsection 4.1.4) to the spreadsheet that aligns and resizes the focal area onto cell boundaries (Figure 4.11). The focal

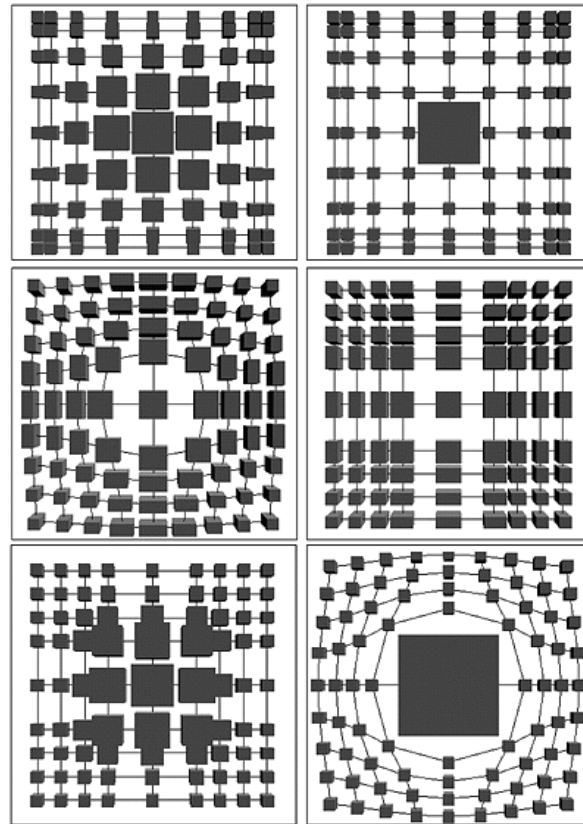


Figure 4.10: Rarer distortion patterns (adapted from Carpendale et al., 1997a)

area can be manipulated with the three standard operations: *zoom* to change the amount of space allocated to the focal area without changing the number of cells contained within, *adjust* to change the number of cells within the focal area, and *slide* to move the focus to contain different cells.

The Table Lens uses different types of graphical representations to display cells depending on each cell's current region (focal, column focal, row focal, or non-focal) and size. For example, a cell containing a numeric value would show the corresponding digits if it has been assigned enough space otherwise it could display a graphic in the allocated space. Even if this graphic is only one pixel high it will still be able to give users a “feeling” for the data, especially relative to neighbouring cells. This is a form of semantic zooming in that each cell adapts its representation to its allotted space.

Year	Product		Quarter	Channel	Units	Revenue	Profits
1993	ForeCode Pro						
1992	ForeWord Pro	539	1	VAR	1	226	79
		540	1	Retail	16	3200	961
		541	1	Retail	12	2400	720
		542	1	Retail	5	1000	300
	ForeMost Server						
	ForeMost Lite						
	ForeMost Access	756	4	VAR	761	684900	287658
		757	4	VAR	475	427500	179550
		758	4	VAR	428	385200	161784

Figure 4.11: Table Lens (from www.parc.xerox.com)

4.1.7 Hyperbolic Display

The hyperbolic plane (Lamping and Rao, 1994; Lamping et al., 1995; Lamping and Rao, 1996) is a non-Euclidean geometry in which parallel lines diverge away from each other. This gives the property that exponentially more space is available as the distance from the centre on the plane increases. Hierarchies or trees, which tend to expand exponentially with depth, can be laid out on a hyperbolic space so that the distance between parents, children and siblings is approximately the same everywhere in the hierarchy. The hyperbolic plane is then mapped, using the Poincaré model, onto a two-dimensional disk with one point, the current focus, in the centre and the rest of the plane fading off in a perspective-like fashion towards the edge of the disk (Figure 4.12). Once mapped onto the display the amount of space available to a node decreases as a continuous function of its distance from the focus node. A typical display has enough space to show several levels of the hierarchy with each level getting less space as it is further from the centre. This gives a degree of context around the focal node. Changes of focus (moving a different node into the centre) are animated and performed in such a way that the parent of the focus node is in a fixed direction from the focus. Animation speed is maintained by drawing intermediate steps with less accuracy; lines are drawn instead of arcs and less of the fringe is drawn.

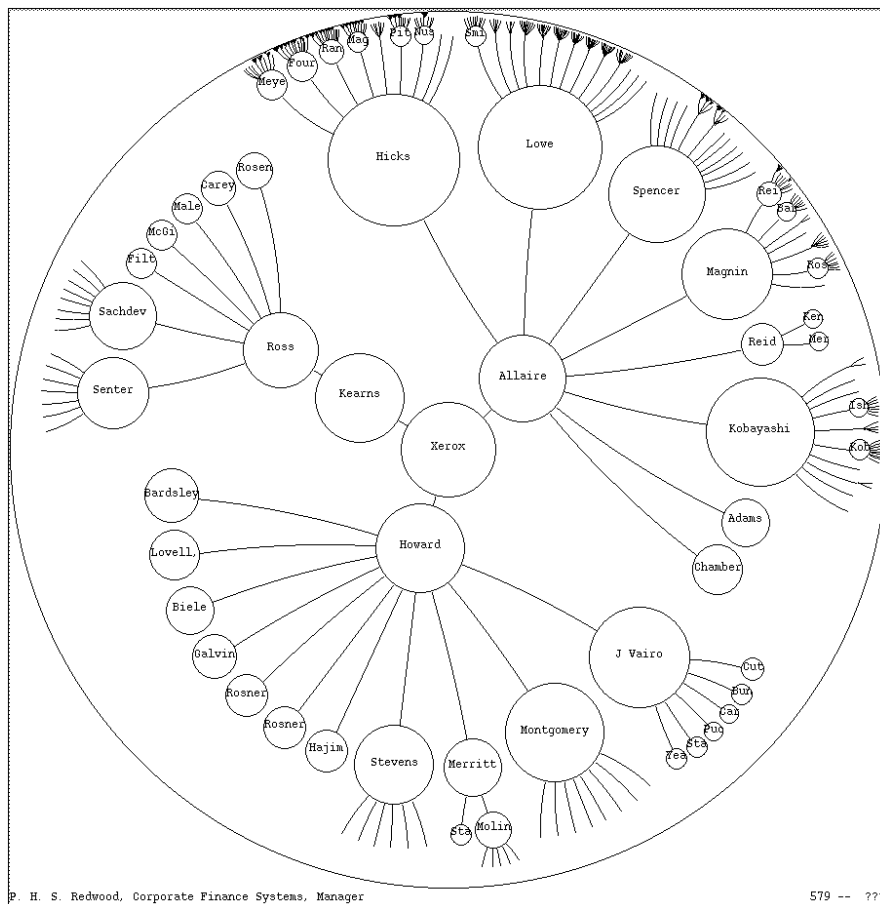


Figure 4.12: Hyperbolic display (adapted from Lamping et al., 1995)



Figure 4.13: Linear magnification (adapted from Keahey and Robertson, 1996a)

Lamping and Rao (1996) also present modifications to the mapping function which allow two focal points or the possibility for users to place more nodes near the focus.

4.1.8 Linear and Non-Linear Transformations

Keahey and Robertson (1996b) propose methods for implementing general non-linear magnification transformations required by fisheye views, hyperbolic planes and 3D pliable surfaces.

Keahey and Robertson state that linear transformations, that is to say constant magnification across a part of the information space, are the simplest for users to understand. These transformations are similar to what users are used to seeing with a magnifying glass. A disadvantage is that users have to understand the abrupt changes in magnification. Also, if the magnified image is in a separate window then users have two transitions to understand: the different magnification levels and the relationship between the two windows. If the magnified image is in the same window as the main one then some of the main view will be hidden. As the magnified view is often drawn on top of the area being magnified (and is bigger than that area) some of the main image is hidden. The hidden area is that immediately surrounding the focus and it is this part of the image that provides the context for the focus. The context is thus lost. Figure 4.13 illustrates the problem. Users are interested in Boris Yeltsin's facial expressions and have thus magnified his nose. Now the users can no longer see the rest of his face.

Non-linear transformations such as those used by fisheye views, hyperbolic

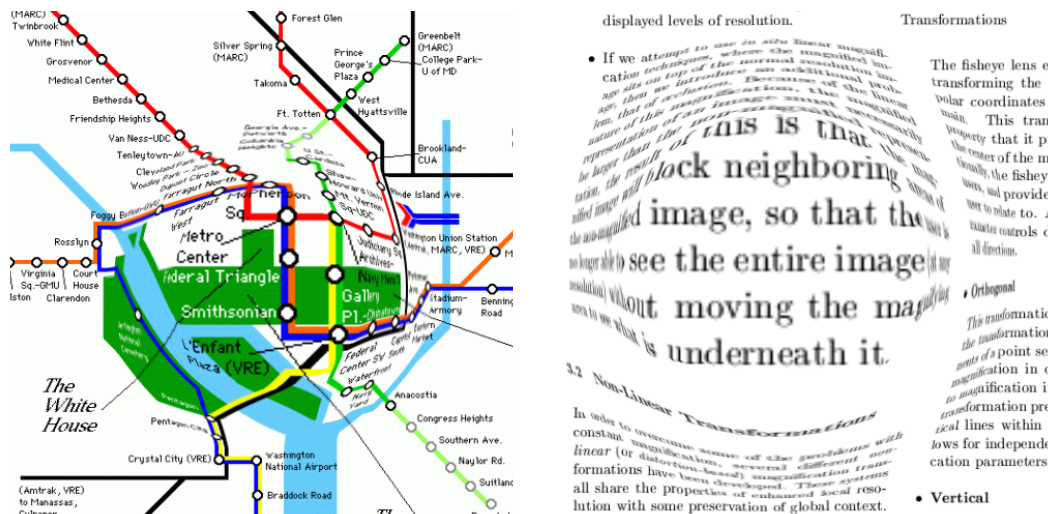


Figure 4.14: Linear and non-linear magnification (adapted from Keahey and Robertson, 1996a)

displays and 3D pliable surfaces avoid the discontinuities of linear transformations by a single (complex) transformation function to the information space. The disadvantage here is that all of the information space is distorted, even the focus of attention, and so nothing in the space can be viewed correctly.

Keahey and Robertson thus suggest combining linear and non-linear transformations. With this combination, the focus of the attention is presented non-distorted (and magnified). Those points around the focus of attention are transformed with a non-linear transformation such that they fit into the remaining space. Figure 4.14 shows views of two information spaces with the focus in the centre. The focus is readable, as is the information far from the focus (that on the edges on the images). The problem is that the information immediately surrounding the focus (often the most important information for understanding the focus) is severely distorted. This information is not hidden as it would be with a linear transformation in the same window but it may be unreadable.

4.2 Visualizing Hierarchies

The standard method of representing a hierarchy (or a tree) flattens the tree into a one dimensional form. The other (horizontal) dimension is used to indicate the depth of a node. Icons are used to represent the nodes in the tree and the links which structure the nodes by straight lines. Users can open and close nodes by clicking on them. A typical file system browser is shown in Figure 4.15. These browsers use logical deformation. Closed parts of the tree do not completely

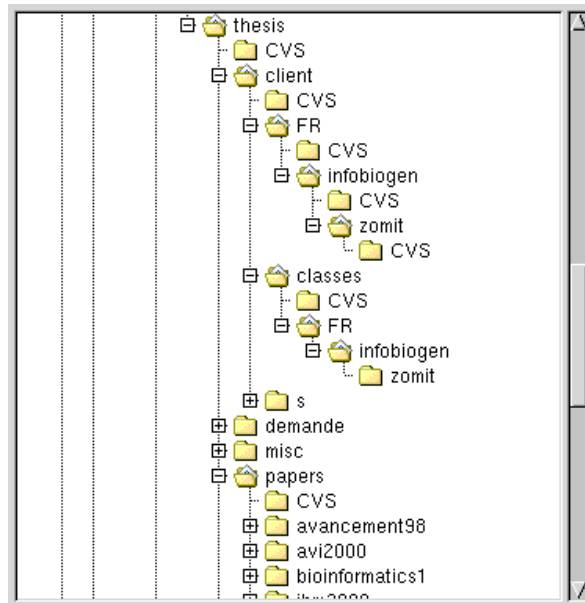


Figure 4.15: Directory browser from Xmms

vanish but are represented by the name of the directory and a plus sign indicating that files have been hidden. A problem with this system is that a large amount of space is wasted in the horizontal dimension. A second problem is that if the tree being visualized is too big for the screen then the context is lost as the user descends in the tree.

The different visualization techniques summarised in this section provide different responses to these problems and illustrate the use of the taxonomies presented in the previous section for displaying a particular type of data.

4.2.1 gIBIS

The gIBIS system (Conklin and Begeman, 1988) converts a directed graph into a tree by following a principal link. Two views of the tree are provided. The first, shown on the left hand side of Figure 4.16, is a global view of the tree. The second, in the upper right of the figure, is a flattened view of the entire tree. This second view (also shown in Figure 4.17) is created by a depth-first traversal of the tree. Nodes can be selected either through the main window or via this index. This index is a second browsing method for the tree. The two windows are synchronised when a node is selected in one window, the other window scrolls so as to also display this node. This is a multi-window (space multiplexed) interface where each window is scrollable (time multiplexed).

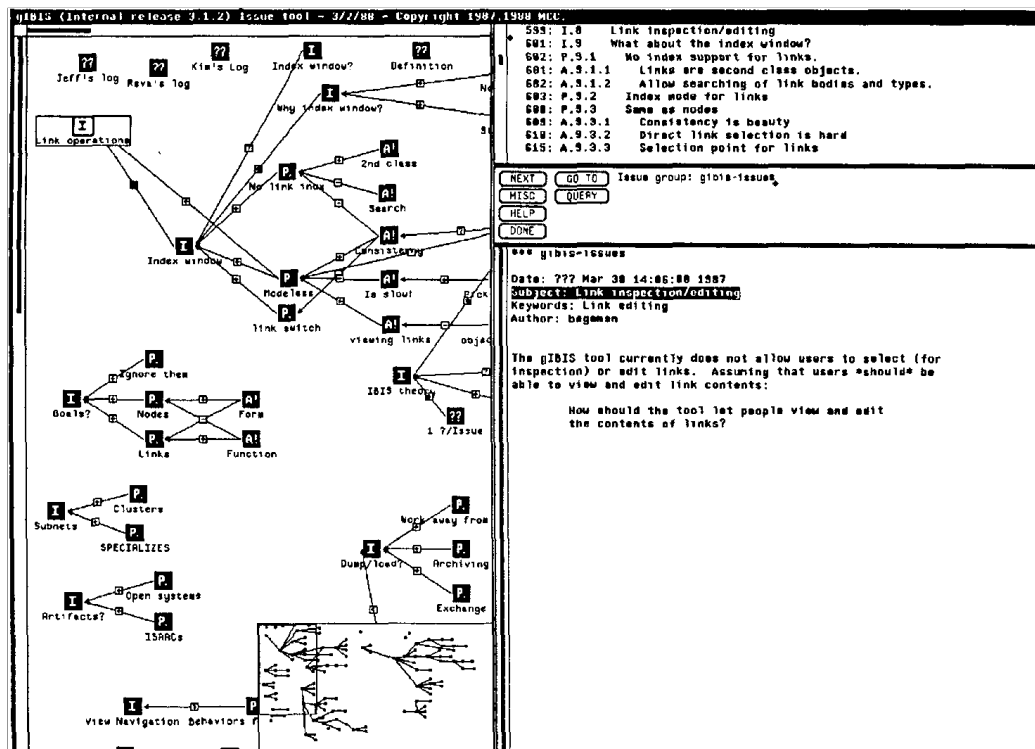


Figure 4.16: gIBIS interface (adapted from Conklin and Begeman, 1988)

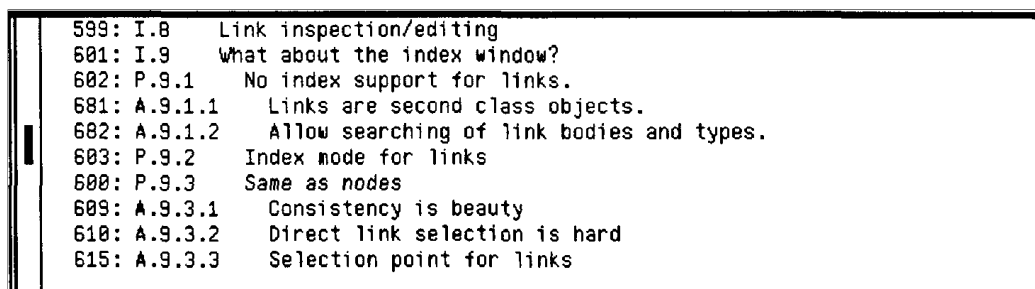


Figure 4.17: gIBIS node index window (adapted from Conklin and Begeman, 1988)

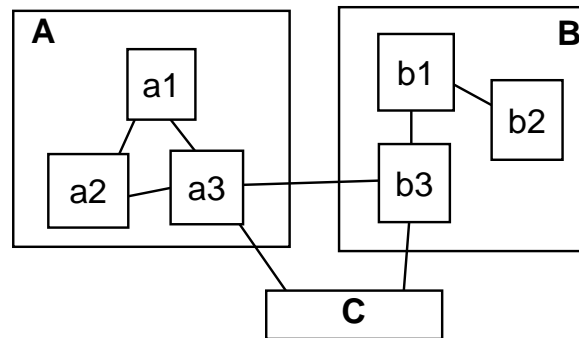


Figure 4.18: Clustered hierarchies with 2 open clusters (adapted from Bartram et al., 1995)

4.2.2 Clustered Hierarchies

Bartram et al. (1995) propose a system for viewing clustered hierarchies (essentially trees) where the display space is managed by recursively breaking it up into smaller rectangular areas, creating a hierarchy of nested rectangles. Each interior node of the hierarchy can be either open or closed. If it is closed then it is represented in a summary form; none of its sub-nodes are visible. An open interior node shows its sub-nodes, which can be either open or closed, and is allocated more space than a closed node (Figure 4.18). The allocation of space to nodes is done automatically and is recalculated when the user opens (or closes) nodes or increases (or decreases) the amount of space allocated to a node.

This system is a multi-focus user interface with a continuous fisheye view. It is possible to open a number of nodes, open nodes can have their scales changed by small amounts, and to define many different scales.

This user interface can handle large hierarchies because normally only a small number of nodes are open. Links between nodes (other than those creating the hierarchical structure) are simply drawn as lines between the nodes. As the number of links increases it will become difficult to layout the graph in order not to have too many crossing lines.

Schaffer et al. (1996) present a system similar to that proposed by Bartram et al. (1995) and cite an experiment to validate the advantage of their system (2D progressive exposure of hierarchical detail combined with fisheye space allocation and multiple focal points) over a full-zoom system. Students were asked to navigate through a telephone network to find and “fix” a broken telephone line. Each student used both the fisheye and full-zoom systems and the researchers found that the students completed the task much faster when using fisheye views.

Schaffer et al. (1996) also installed a fisheye interface in an electricity utility’s control room (Figure 4.19). The operators preferred the fisheye system but had

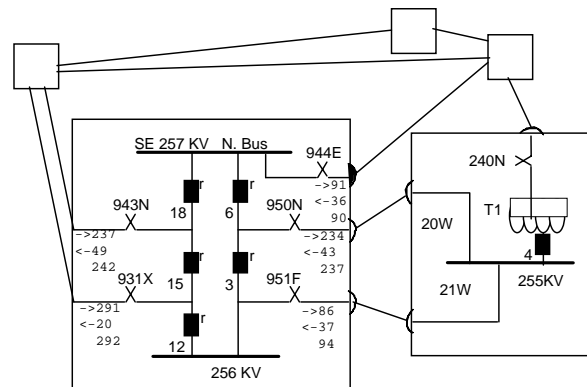


Figure 4.19: Two nodes expanded to circuit diagram level (adapted from Schaffer et al., 1996)

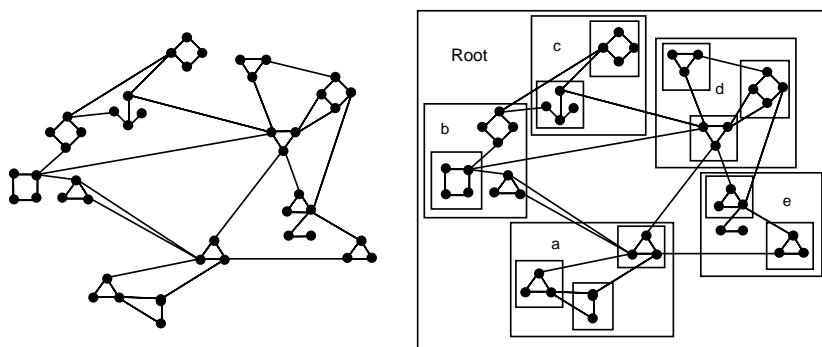


Figure 4.20: Hierarchically clustering a network (adapted from Schaffer et al., 1996)

reservations about the screen space lost to the focus to show the context when at the bottom level of the system. A toggle to remove the context or a control to adjust the balance between context and focus could be the solution to this problem. They also found the same problem as Bartram et al. (1995): how to represent links (which break the hierarchical structure of the network) between nodes. Figure 4.20 shows how a network can be hierarchically clustered. A later version of their system will use animation to aid the user in following changes to the layout and perhaps allow flexibility in the choice of nodes to display when a cluster is open. Not all nodes are necessarily interesting for a given task.

It is also suggested that motion (for example, vibration) might be better than a colour change at attracting attention to an important region such as a trouble spot.



Figure 4.21: ZoomTree showing local focus+context views (thanks to Laurent Robert)

4.2.3 ZoomTree

ZoomTree (Robert and Lecolinet, 1998, 2001) is a system for browsing one or more hyperdocuments (in the form of trees) that shows multiple focal points of varying importance while maintaining the context of these foci. Having chosen the hyperdocuments of interest and their root nodes (or pages), users are shown a representation of the trees with all the pages shown at the same size. Users can enlarge (zoom) pages of interest in four different ways. The first is hierarchical zooming: a selected window and all its children are zoomed. The second is contextual zooming, which resizes windows as a function of their relationship with the selected page. Pages are given an importance that is calculated as the inverse of the number of links from the selected page. Important pages grow while unimportant pages shrink. The third form of zooming grows or shrinks all the visible pages by the same proportion. The fourth way of zooming a page is to click on a link in another page that refers to it. (Clicking on a link to a page that is not yet visible will cause it to be loaded into the browser.) Figure 4.21 shows a ZoomTree where the users have zoomed on the centre page in order to see it in detail. A number of other pages are also interesting and have been enlarged so that their contents are more readable.

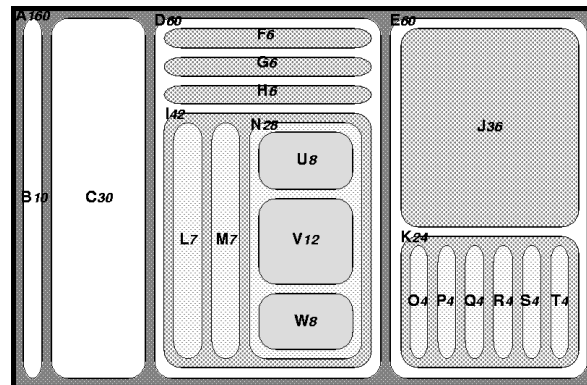


Figure 4.22: Treemap (adapted from Johnson and Shneiderman, 1991)

4.2.4 Treemaps

Johnson and Shneiderman (1991) propose two static methods for presenting large hierarchical information spaces (trees) by tiling two-dimensional rectangles (Turo and Johnson, 1992). The algorithms assign the entire plane to the root node and then divide the root node's area among its children in proportion to each child's weight compared to the sum of the children's weights. The algorithm is then repeated for each of the non-leaf children. The weights can be any attribute of the underlying objects interesting to the user. One example is the size of files in a hierarchical file system but it could also be the sales results of sales people in a hierarchically organised company.

The first algorithm, “top-down,” starts at the either the top or the left of the display and continues in the same direction until the areas assigned become too narrow. This happens rapidly because displays have only a limited width or height. An alternative algorithm changes the direction of the subdivision of the display at each recursion. This means that the regions become narrower and shorter rather than just narrower (Figure 4.22). Instead of being 100×1 pixels, a region might be 10×10 pixels, making selection and labelling easier.

Zooming was implemented and allowed the user replace the entire display with the contents of a single node. Animation was used to show what was being zoomed in to or away from but no context was provided to stop users getting lost.

TennisViewer (Jin and Banks, 1997) is a browser for the results of a tennis match using Treemaps and Magic Lenses (subsection 4.3.6). The top level of the Treemap is the match; and the bottom level contains graphical representations of the points. Magic Lenses can be used to zoom on an area.

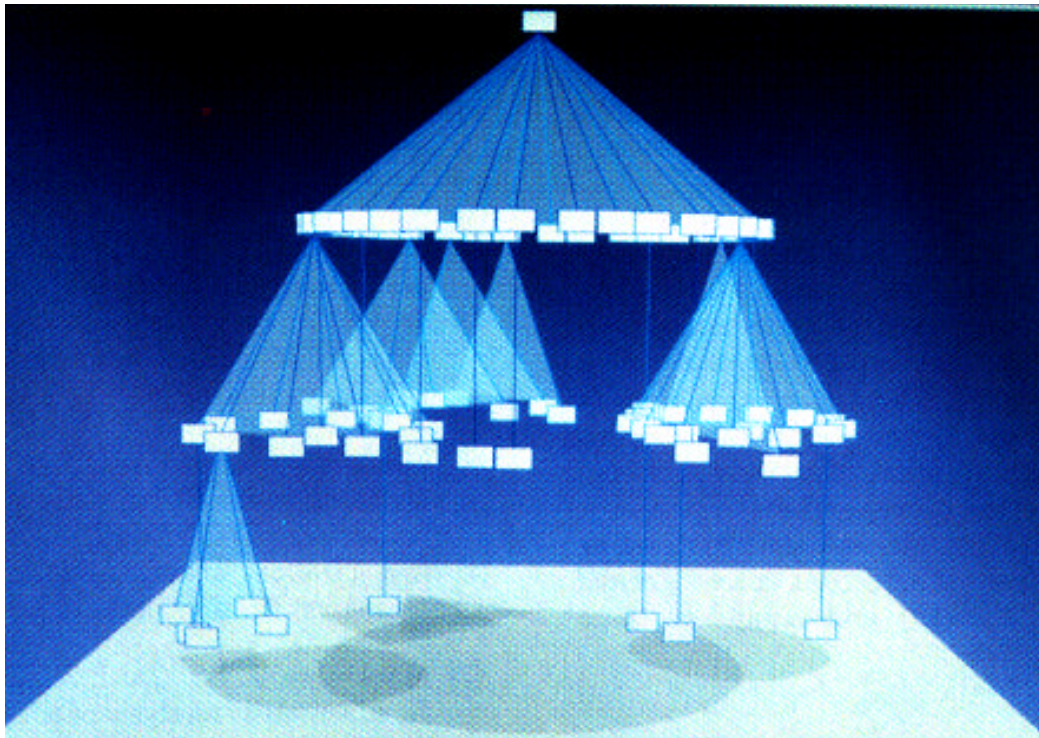


Figure 4.23: Cone Tree (adapted from Robertson et al., 1991)

4.2.5 Cone and Cam Trees

Robertson et al. (1991) presents Cone Trees, three dimensional hierarchical representations of trees (Figure 4.23). Three dimensional representations use an optical deformation (a projection from three dimensional space onto a two dimensional computer screen) that users are meant to assimilate and eliminate, thus mentally recreating the three dimensional image. The disadvantages of this type of representation come from the problems of occlusion and the difficulties in navigating in a three dimensional space with a two dimensional pointing device (mouse).

Most trees in real applications tend to be broad and shallow. This aspect ratio makes trees hard to fit into two dimensional layouts while three dimensional Cone Trees use depth to fill the screen with more information. The node at the top of the hierarchy is placed near the top of the screen and is the apex of a cone. The top node's children are equally placed around the circumference of the first cone and further cones drawn. All cones are the same height: the height of the screen divided by the tree depth. The entire tree is thus always visible and the cones transparent so that they can be seen but do not block the view of the cones at the back. Cone Trees can be pruned (all descendents of a node hidden) and rotated by the user. All tree modifications are animated and take about a second to help the

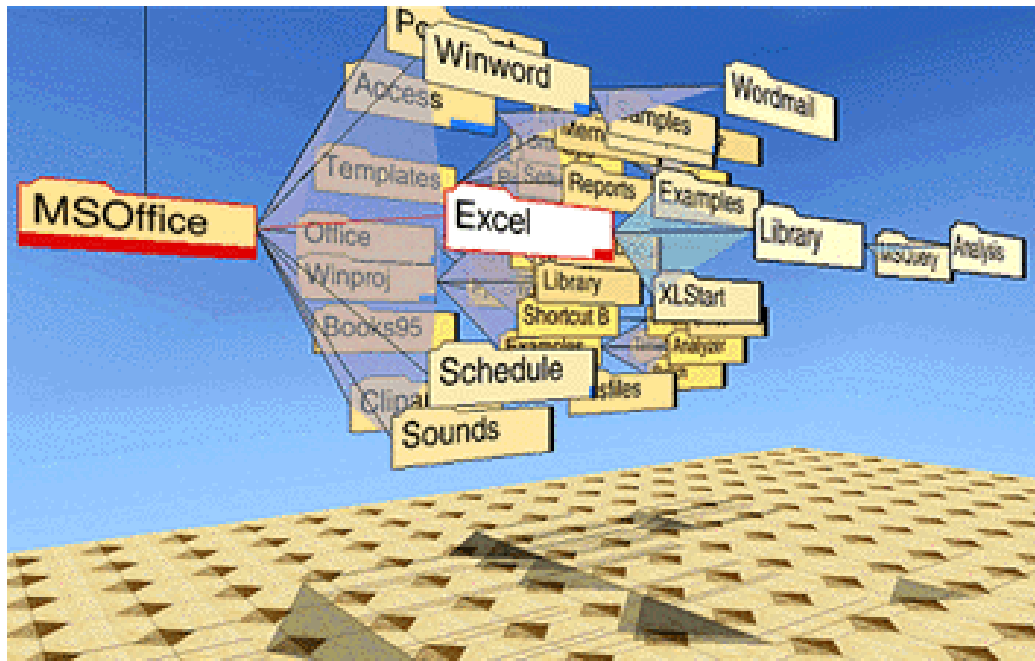


Figure 4.24: Cam Tree (from www.parc.xerox.com)

user track the changes. The data underlying the trees can be searched. At the end of the search nodes are highlighted with a red bar whose size indicates its relative search score.

Cone Trees also provide a fisheye view (subsection 4.1.1) without having to define a degree of interest function. The selected path in the tree is automatically brighter, closer and larger than other paths, due to the three dimensional perspective view, colouring and simulated lighting.

Cone Trees have been used in the Information Visualizer (Robertson et al., 1993) to display Unix file systems, organisational structures, and company operating plans.

The paper also discusses Cam Trees (Figure 4.24). These are very similar to Cone Trees except that they are drawn horizontally.

Cone and Cam Trees are less effective with trees containing more than 1000 nodes, 10 layers or a branching factor greater than 30. They are more effective with unbalanced trees because the differences in structure make the trees easier to track when rotated.

4.2.6 Multitrees

Furnas and Zacks (1994) state that hypertext systems are frequently general graphs and that such graphs are powerful because they allow many routes between nodes

	space	focus change	strategy	deformation
directory tree	1D	scroll & icons	time	logical
Treemap	2D	zoom	time	position
Cone Trees	3D	rotate	time & depth	pseudo-optic
gIBIS	1 & 2D	scroll	space & time	logical
clustered	2D	pruning	time	logical
ZoomTree	2D	zoom	time	pseudo-optic
Multitrees	2D	change node	time	logical

Table 4.1: Simple tree visualization taxonomy

(for, for example, cross references and multiple organisations) they are however very difficult to layout and difficult for the user to understand. Trees are much simpler structures: they can be easily laid out in a plane, they can be completely traversed easily, and are thus easy to understand. Trees are limited because there can only be one route between objects (thus prohibiting shortcuts, cross references and alternative organisations). Directed acyclic graphs (DAGs) are somewhere between trees and general graphs, they support top down search strategies and a natural orientation like trees, but they can, like general graphs, be difficult to lay out and understand.

Multitrees are a generalisation of trees but are still more restrictive than DAGs. A multitree is a DAG whose nodes have descendents forming only trees and is essentially a structure containing many trees. Each of these trees is a different use of the data in the structure (Figure 4.25). A multitree can be more easily browsed than a DAG because at each node there are only two trees to be laid out, one a tree of descendants and the other a tree of ancestors. As the user moves through the tree the context of the current focus (the ancestors and the descendants of the current node) can always easily be shown by two trees (Figure 4.25).

4.2.7 Discussion

Table 4.1 shows a simple way of classifying the tree visualization techniques presented in this chapter. The most suitable technique depends on the size of the tree relative to the available space. If space is at a premium then the more space-efficient two or even three dimensional display should be used. A further consideration is the choice of deformation technique.

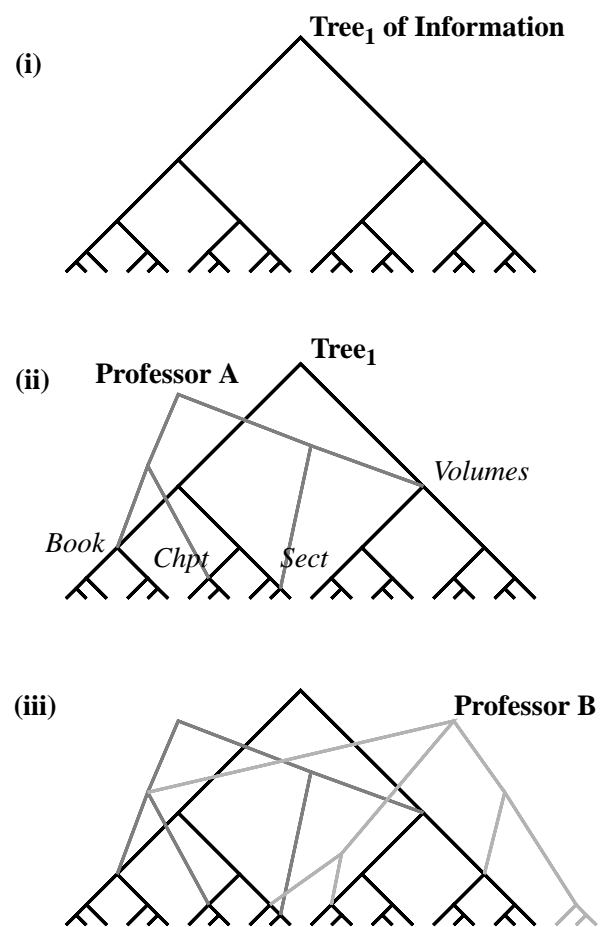


Figure 4.25: Multitree showing an information tree used by two professors (adapted from Furnas and Zacks, 1994)

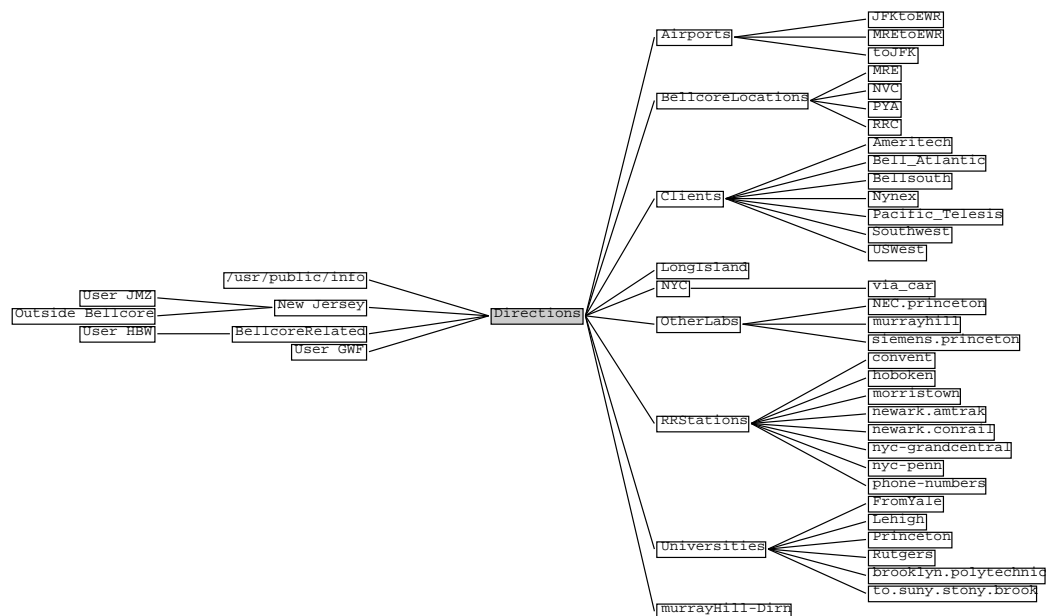


Figure 4.26: Centrifugal view centred on the node “directions” (adapted from Furnas and Zacks, 1994)

4.3 Transparent Tools

Transparent tools are small temporary screen overlays that users can create, manipulate and destroy. They include menus, cursors, Toolglasses and Magic Lenses.

4.3.1 Introduction

The limited screen size and the increasing numbers of windows, menus, dialog boxes and tool palettes has lead to screens becoming increasingly cluttered. Traditional tiling of these objects is unwieldy because of the number of objects and because tiling often leads to important information (or windows) being hidden. Dialog, help and menu popups block part of the main window and, since they pop up near the cursor, often the part of the window that the user would most like to see. (If they do not pop up near the cursor then users might not see them and wonder why their program has blocked.) Kurtenbach et al. (1997) state that making the screen bigger is not a good solution because with huge screens users spend too long moving from one side of the display to the other, and then refocusing on their task. Screens have also grown to almost their maximum size given the place available on desks. Another aim of user interface research is to create new techniques suitable for the relatively very small screens of portable devices such as electronic organisers, Pocket PCs, and mobile phones.

Depth Mutliplexed Strategy

Harrison et al. (1995a) attempt to transform the usual space multiplexed or tiled strategy into a depth or layered multiplexed strategy. These two strategies are the extremes of a continuum from fully opaque (traditional) to fully transparent and can be combined in a way optimised for each task. Fully transparent displays are found in aviation's Heads Up Displays. Using more advanced hardware an extra dimension can be used: the depth of planes. Not only can objects be transparent, they can also appear to be closer to the user. Each object can be anywhere between opaque and transparent. An important object should be more opaque than an object that is currently relatively uninteresting to the user. Research has shown that when users are presented with two transparent overlaid displays that they are capable of ignoring one display and concentrating on the other.

Distinguishing Layers

There are many different ways of distinguishing layers, such as different colours, different graphical or visual forms (text versus graphical window contents), font size or styles, layer movement (Belge et al., 1993), layer jitter (Silvers, 1995), camera movement and stereo. Semi-transparent objects can be made more legible by "anti-interference" techniques, such as by outlining text and other objects with a contrasting colour (black text is surrounded by a one pixel wide white outline).

Approaching Objects

The frequent need to focus attention on several items requires users to move their attention between different screen objects in different region of the screen. Semi-transparency will allow the user to place several different objects in the centre of the workspace without some of them obscuring the others.

Integrating Task and Tool Space

Harrison and Vicente (1996) state that transparency is perhaps most useful in better integrating task space and tool space. Many applications have a large work space, which is the primary focus of attention, and tools to manipulate that work space. These tools overlay, hide, and distract from the main work space. The authors experimented with semi-transparent text menus on top of images in an attempt to find which fonts and transparency levels were legible while leaving as much of the main image as visible as possible (Figure 4.27). In these experiments the text was always opaque, it was the transparency of the surface surrounding the text that was varied.

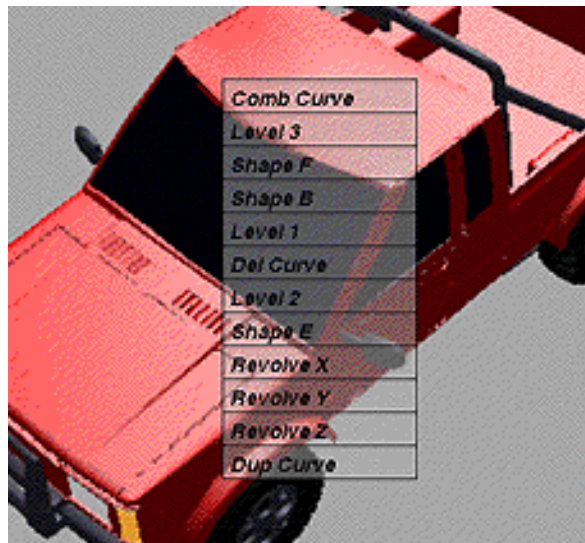


Figure 4.27: Semi-transparent text menu (adapted from Harrison and Vicente, 1996)

Kurtenbach et al. (1997) use transparency, marking menus (subsection 2.1.4), Toolglasses (subsection 4.3.5), and other techniques in a program for creating simple 2D graphics.

4.3.2 Semi-Transparent Tool Palettes

Harrison et al. (1995b) evaluated the usability of semi-transparent tool palettes in a technical drawing application (Figure 4.28). Three different types of icons were used: text, line art and solid rendered objects. Palettes were constructed randomly, removing the possibility to learn the positions of frequency used icons. Three different type of background images were used: text pages, wire frame images, and solid images. Different levels of transparency, from 0% (opaque) to 90%, were assigned to the palette.

Transparency levels between 0% and 50% were found to work well. Solid backgrounds were easiest to read, followed by text and wire-frame backgrounds. Solid icons were the easiest to use, followed, as for the backgrounds, by text then line art.

These tests were short term. As users learn which features distinguish icons that they use frequently recognition could be improved and errors reduced even with very transparent icons. As users start to know where frequently used icons are positioned they prefer more transparent palettes and thus an increased view of the underlying image. The transparency of the palettes should adapt to the



Figure 4.28: Semi-transparent tool palette (adapted from Harrison et al., 1995b)

experience of the user and icons adapted for use in a semi-transparent palette.

4.3.3 Semi-Transparent 3D Cursors

Zhai et al. (1994) use a semi-transparent 3D cursor to acquire targets in a 3D environment. The cursor, called a “Silk Cursor”, is a volume rather than a point and its surfaces are semi-transparent or made out of “silk”. Objects are seen differently depending on the number of layers of silk between them and the user. Objects can thus be seen to be in front of, in, or behind the cursor. Their experiments show that volume and occlusion cues are useful in both monocular and stereoscopic conditions. Users were able to acquire 3D targets more easily with a silk cursor than with comparable wire frame cursors.

Figure 4.29 shows a fish partially inside a semi-transparent 3D cursor. The eye and part of a fin are in front of the cursor, the top and bottom fins and the top of the tail are behind the cursor, and the body is inside.

4.3.4 Translucent Patches

Translucent Patches (Kramer, 1994, 1996) are translucent (semi-transparent), irregular, user controlled windows. The central idea is to allow the user to make freehand sketches on a window in a patch and then to apply interpretations to the patch (Figure 4.30). Instead of choosing an application and then “filling in the blanks”, this approach puts representation before structure and interpretation. Any mark made on the screen is valid; meaning (from the computer’s point of views) is assigned to the marks when and if necessary.

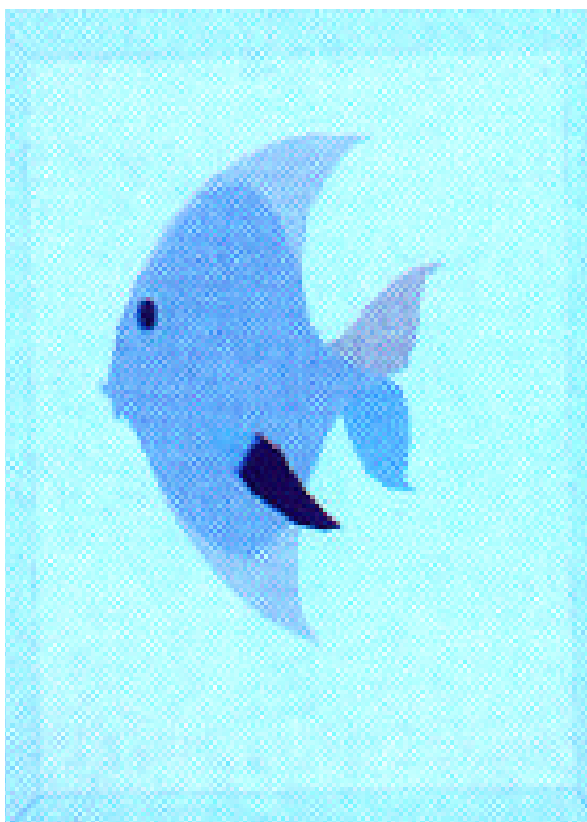


Figure 4.29: Semi-transparent 3D cursor (adapted from Zhai et al., 1994)

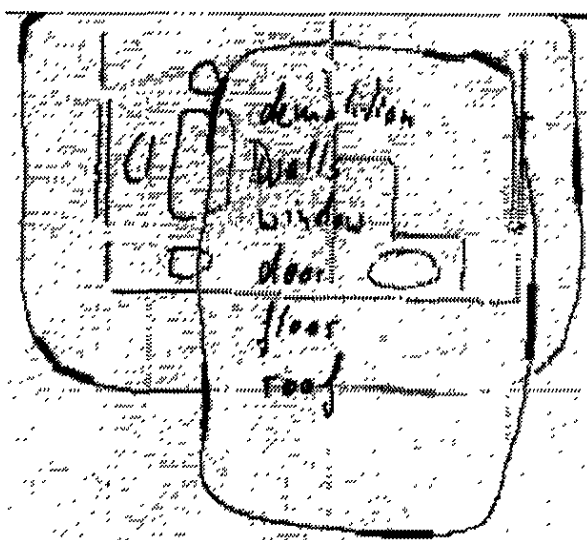


Figure 4.30: Translucent Patch containing a list drawn over a sketch (adapted from Kramer, 1994)

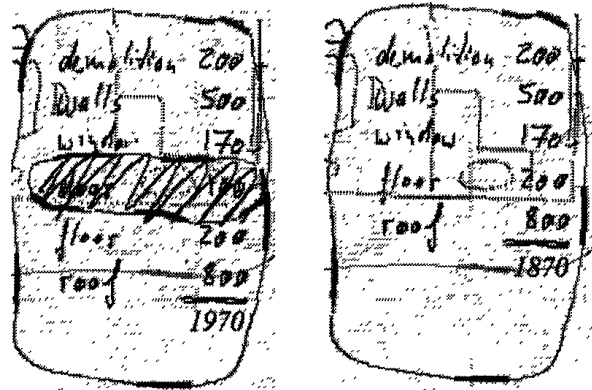


Figure 4.31: Calculator applied to a Translucent Patch (adapted from Kramer, 1994)

For example, numbers could be drawn in a patch and then a calculator interpretation applied to the patch to produce the sum of the numbers (Figure 4.31). This patch would be transparent and thus the numbers could be drawn next to the objects that they concern without those objects being hidden. The fact that these patches are semi-transparent means that the user can focus, temporarily, on one part of the display, the numbers, while still keeping the wider context visible. In addition as the patches are temporary they can be used to experiment with a document without modifying it directly. Patches have an order, they are stacked one above the other, and can thus be used to record or represent a history of changes, possibly by different authors, to a document. A patch can also be removed without destroying it. This allows the user to see the document without a patch without losing the patch.

A set of gestures to control patches are proposed. For example, a patch can be emptied by drawing a zig zag that fills the patch, or, a patch can be deleted by drawing a zig zag that crosses the patch's border.

4.3.5 Toolglass Widgets

Patches (subsection 4.3.4) can omit some of their details when they are covered by other patches, and, conversely, patches can modify patches that they cover. In this sense they resemble the Toolglass™ widgets proposed by Bier et al. (1993). Toolglass widgets are transparent and are positioned with the non-dominant hand on an invisible sheet between the application and the cursor. Each Toolglass widget provides a function which is applied to the application object by clicking through the Toolglass with the cursor. One function is colour selection; the user clicks through the part of the Toolglass widget with a red corner tab to change the colour of the

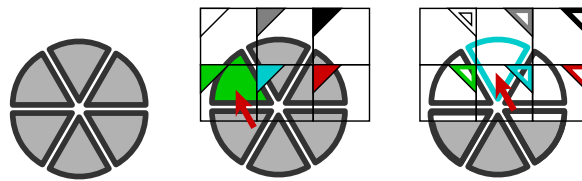


Figure 4.32: Colour changing Toolglass (adapted from Bier et al., 1993)



Figure 4.33: Creating and positioning objects with a Toolglass (adapted from Bier et al., 1993)

object under the cursor to red (the middle image in Figure 4.32). A similar operation is changing the outline colour of an object (the right image in Figure 4.32). Here the tabs in the Toolglass indicate that it is the outline that is to be changed. In addition, only the outlines of those objects covered by Toolglass are drawn. This makes the function of the Toolglass more obvious and the current and new colours easier to see.

Another example is the creation of shapes. In Figure 4.33 the user “pushes” a circle from the shape Toolglass onto the application screen. The user can then position the new shape before releasing the mouse button.

Users can use Toolglasses to create macros (visual programming) as overlapping Toolglasses combine their effects. Clicking through the two Toolglasses just discussed will create a red circle. A Toolglass that creates red circles can be made by overlapping and welding together these two Toolglasses.

A single Toolglass can also be used in different types of applications. A Toolglass that makes graphical objects go red in a painting program can make words go red in a word processor.

4.3.6 Magic Lenses

Magic Lenses from Xerox PARC are (usually) rectangular regions which provide visual transformations of the covered area (Stone et al., 1994). Magic Lenses can provide transformations on the region covered such as activating alignment aids (turning on a grid), modifying the way a picture is rendered on the screen (wireframe versus shaded), and filtering (removing) data. A simple example of a

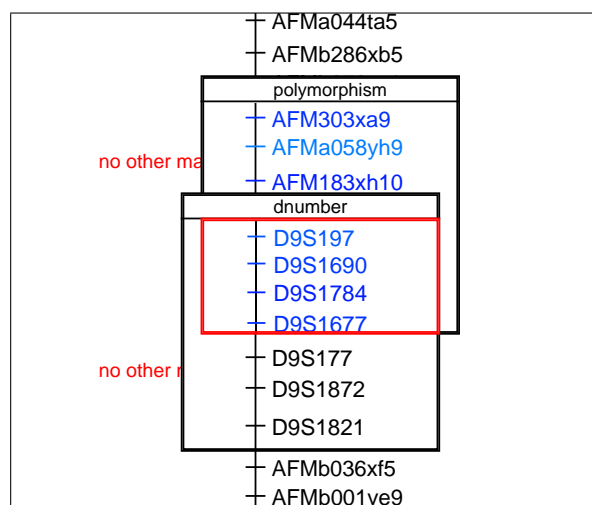


Figure 4.34: Overlapping Magic Lenses

Magic Lens is a rectangle which converts a table of numbers into the corresponding scatter plot. An example of possible use of lenses in molecular biology data visualization is given in Robinson and Flores (1997).

Magic Lenses (and Toolglasses) create spatial modes that can replace the standard temporal modes (where the user changes the state of the application with, for example, a menu) provided in user interfaces.

Combining Magic Lenses

In the same way that Toolglass widgets (subsection 4.3.5) can be combined, Magic Lenses can be combined temporarily or welded together to provide more complicated transformations. Figure 4.34 from ZoomMap (section 7.1) shows two Magic Lenses. One, called “polymorphism”, uses the colour of objects (the shade of blue in this case) to indicate the value of a normally hidden parameter of the object. The second, called “dnumber”, shows an alternative name of the object. Where the two overlap the alternative name with the indication of the value of the parameter’s value is shown. The combination of Magic Lenses can be viewed as the creation of visual macros (Stone et al., 1994) or as a particular and restricted case of visual programming. The visual nature of this technique means that it is a very intuitive way to program, and thus is useful to users who master the concepts underlying the data transformation but are not familiar with programming.

Another example of overlapping Magic Lenses is the city map in Figure 4.35. One of the Magic Lenses highlights the major roads in that part of the map covered by the lens. The other shows the waterways. Where the Magic Lenses overlap both features are highlighted.

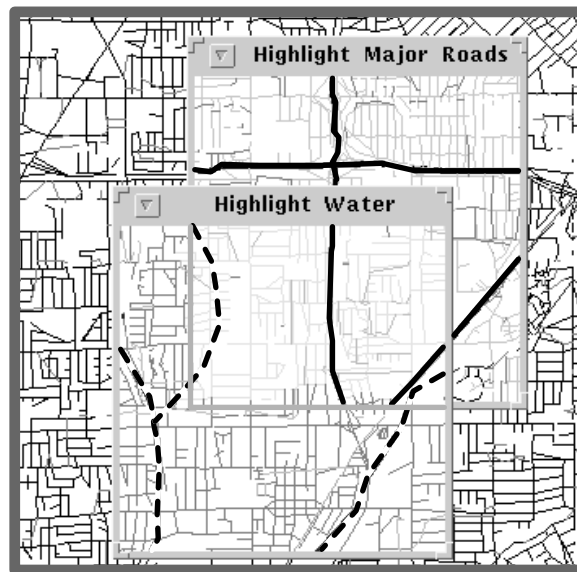


Figure 4.35: Overlapping Magic Lenses on a city map (adapted from Stone et al., 1994)

Magic Lenses that Magnify

A Magic Lens can also be a magnifying glass. This provides additional space in order to display extra information on the objects. The disadvantage of this magnification is that the lens does not transform the entire region covered, but only the objects in the centre of this region. Some of the objects next to the user's focus of interest are invisible. Figure 4.36 shows a Magic Lens, called “info”, that gives extra information on the objects that it covers. To obtain the space to show the extra details on each object, the lens magnifies the space allocated to each object. The Magic Lens in Figure 4.36b shows detailed information on the objects “AFMb347yh9” and “AFMb073xc1”. Objects surrounding these two objects, such as “AFMa119zg9”, are no longer visible. A possible solution to this problem is to show the contents of the Magic Lens (the magnified image) in a separate window. This avoids hiding the objects around the focus but at the price of requiring users to shift their attention backwards and forwards between two screen areas.

A further disadvantage of magnifying Magic Lenses is that, as the lens is moved, the objects shown in the lens move much faster than the lens itself. If the lens magnifies by a factor of two and the user moves the lens by one pixel, those objects shown in the lens move by two pixels. This magnification of the movement makes positioning the lens difficult and can be distracting for the user.

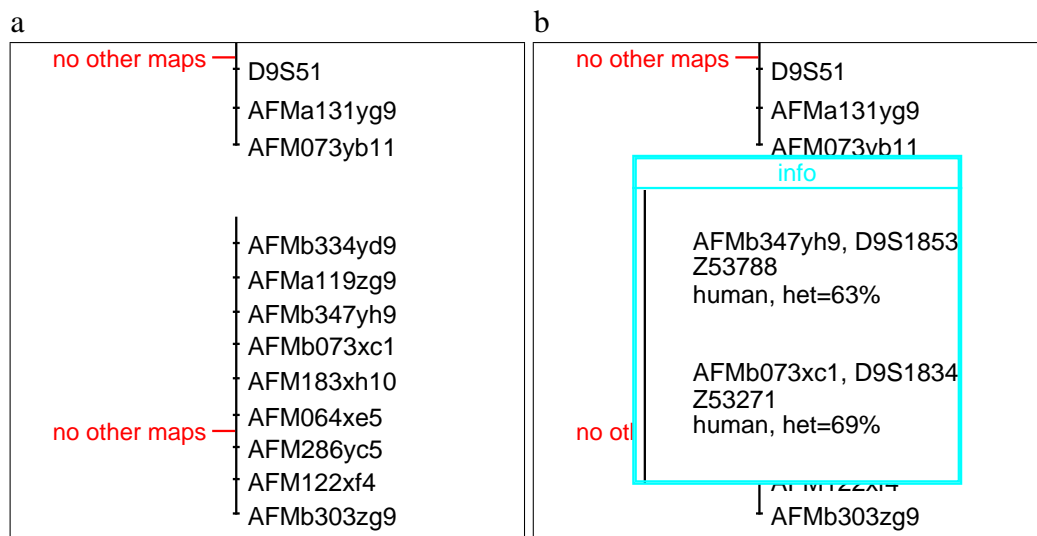


Figure 4.36: Magnifying Magic Lens

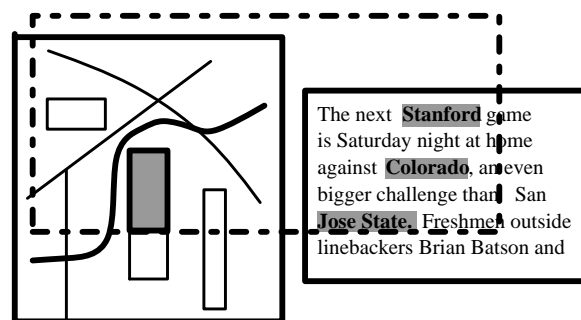


Figure 4.37: Magic Lens in two applications (adapted from Stone et al., 1994)

Multi-Application Magic Lenses

Magic Lenses and Toolglasses can be used across different applications. A lens or Toolglass that changes the graphical properties of objects (the colour for example) can be used in the same way in painting programs, word processors, etc.

Other lenses can have effects that depend on the application on which it is applied. When a Magic Lens that highlights schools is applied to part of document in a word processor it can highlight the names of schools wherever they appear in the text. When used on a map it can indicate the positions of schools. An example of such a lens is shown in Figure 4.37. These lenses allow users to learn one way of obtaining a certain type of information that works across applications.

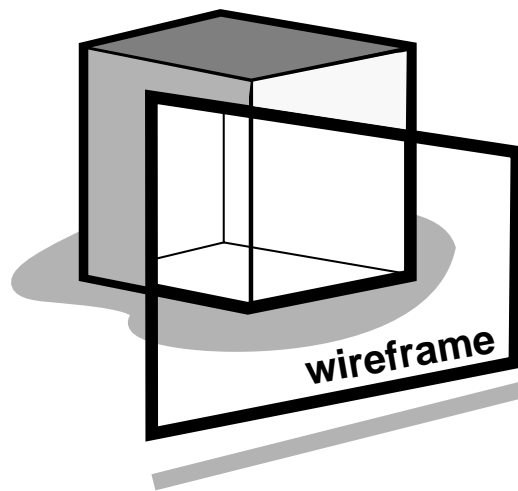


Figure 4.38: Flat lens in 3D space (adapted from Viegas et al., 1996)

Magic Lenses and Toolglasses

Toolglasses and Magic Lenses can be combined. For example, a Toolglass that changes the colours of objects can be combined with a Magic Lens that magnifies to facilitate clicking on the object to be modified.

Magic Lenses in 3D

Viegas et al. (1996) adapt Magic Lenses to 3D environments and propose two new see-through visualization techniques: *flat lenses in 3D* (Figure 4.38) and *volumetric lenses* (Figure 4.39). In a 3D environment a flat lens changes parts of objects; those parts that fall behind the lens from viewer's eye point. Flat lenses can be composed as in a 2D environment, however, if the lenses do not overlap completely, the resulting 3D intersection regions can be complicated and difficult to render with current hardware.

A volumetric lens limits the effects of a transformation to a finite region. This region is fixed, unlike the region transformed by a flat lens, and does not change with the eye point. A volumetric lens could be used to implement, for example, X Ray vision. The covered object disappears, revealing what is inside (the bones inside a hand for example). Volumetric lenses do not have an order of composition. There is no front or back lens. Compositions have to be constructed from the inside out or from the order in which the lenses were applied. Overlapping volumetric lenses can produce many regions in different lens combinations, once again making rendering difficult.

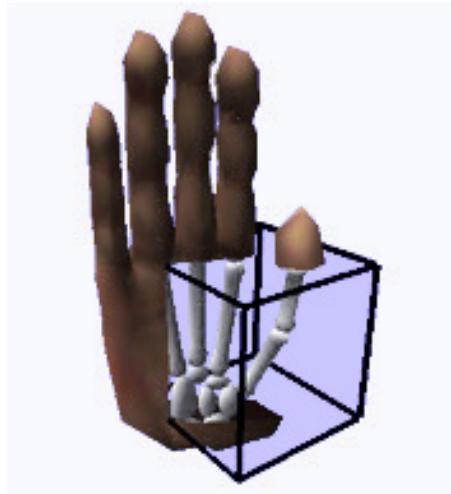


Figure 4.39: Volumetric lens in 3D space (adapted from Viegas et al., 1996)

4.3.7 Transparent Overview Layers

Overviews are global views of an information space. They show all the objects in that space and the position of the user's more detailed view of that space. Overviews can also be active: the objects visible in the overview can be selected and manipulated, and the representation of the user's position moved in order to change the user's position in the detailed view.

The overview is typically displayed in a window separate from that of the detailed view. (The two views can be said to be space multiplexed.) This overview window uses valuable screen real-estate and thus the detailed view must be smaller (and more difficult to use) than with only one view. As the two views are separate users must switch their attention between the two views (or ignore the overview).

Another possibility is to display the overview in the same window as the detailed view and require that the user choose between the two views. (The two views are thus time multiplexed.) This saves screen space but makes it impossible for the user to see the two views at the same time. The user must remember the contents of the invisible view. In addition, switching attention from one view to the other is even more difficult than with the two separate windows as it requires an action by the user.

Cox et al. (1998) propose and evaluate what they call *Transparent Overview Layers*. They seek to avoid the problems associated with space and time multiplexing by showing the overview and the detailed view together in the same window. The overview is drawn on top of the detailed view with the same size as the detailed view. The objects in the overview are drawn transparent so that the contents of both views are visible at the same time. This allows users to identify

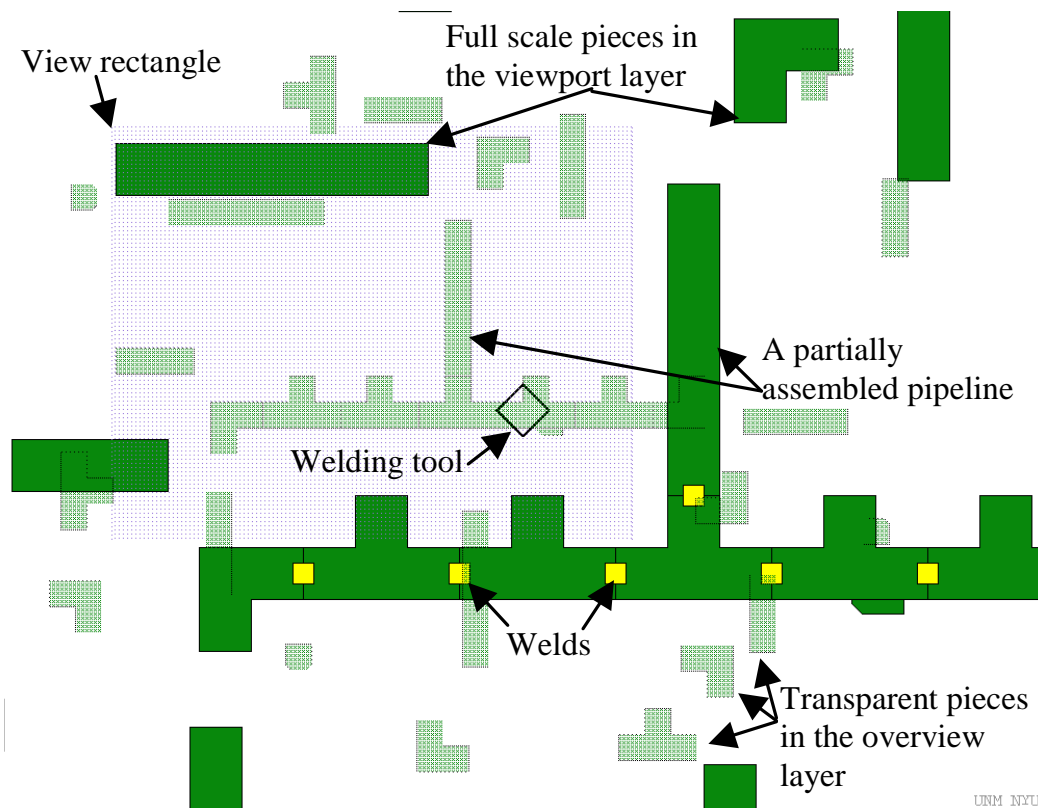


Figure 4.40: Annotated 70% transparent overview layer in a pipeline system (adapted from Cox et al., 1998)

to which view an object belongs. This gives what they call depth multiplexing: the detailed view is “below” the overview. There is however visual interference between the two views.

Figure 4.40 shows a system where users must weld virtual tubes together to create a virtual network of pipelines. The overview shows all the tubes available and the detailed view is a zoomed view of part of the overview. Cox et al. found that users were able to use the overview and the detailed view. Users were also able to easily shift their focus of attention between the two views, even in the middle of an action. Users did however sometimes have problems in distinguishing to which view an object belonged. They sometimes tried to join a tube in the overview to a tube in the detailed view (or visa versa). The frequency with which this happened depended on the level of transparency used in the overview.

Cox et al. tested their system with different levels of transparency (always predefined by the system). Figure 4.40 has the overview 70% transparent. The objects in the overview are thus drawn lighter than in the detailed view. Figure 4.41a shows an object in the detailed view (the solid object) and an object

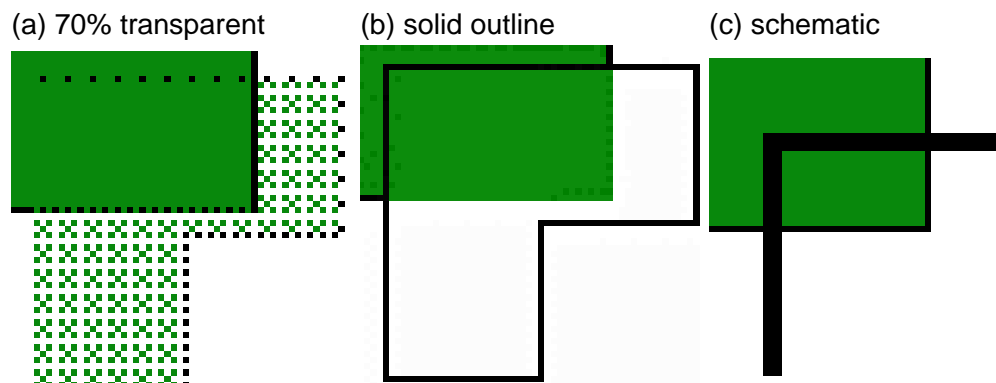


Figure 4.41: Representations of objects used in a transparent overview layer (adapted from Cox et al., 1998)

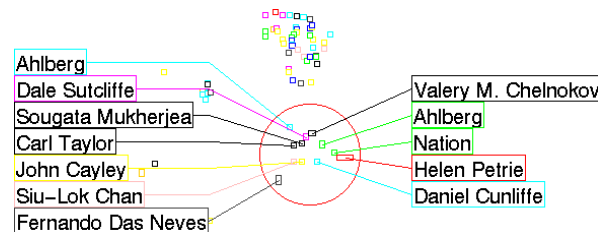


Figure 4.42: Excentric Labelling (from <http://www.cs.umd.edu/hcil/excentric/dist/bin/test7.html>)

in an overview 70% transparent. Figure 4.41b shows the same two objects with the difference that the object in the overview is drawn as a solid outline. In Figure 4.41c the objects in the overview are represented as schematic stick figures. Cox et al. found that users preferred the objects in the overview to be shown as solid outlines.

This system is not a ZUI because the scales of the two views are fixed. The two different scales do however mean that this system has some points in common with ZUIs and in particular with the Macroscopic (subsection 4.4.6).

4.3.8 Excentric Labelling

Excentric Labelling (Fekete and Plaisant, 1999) offers a way of identifying objects on the screen. This technique labels, with “tool tips” in the main view, those objects located around the cursor (Figure 4.42). These labels are only drawn when the user stops moving the cursor. They then remain visible while the cursor is moved slowly and are updated with this movement. The textual labels are distributed around but at a certain distance from the cursor in a regular layout. This

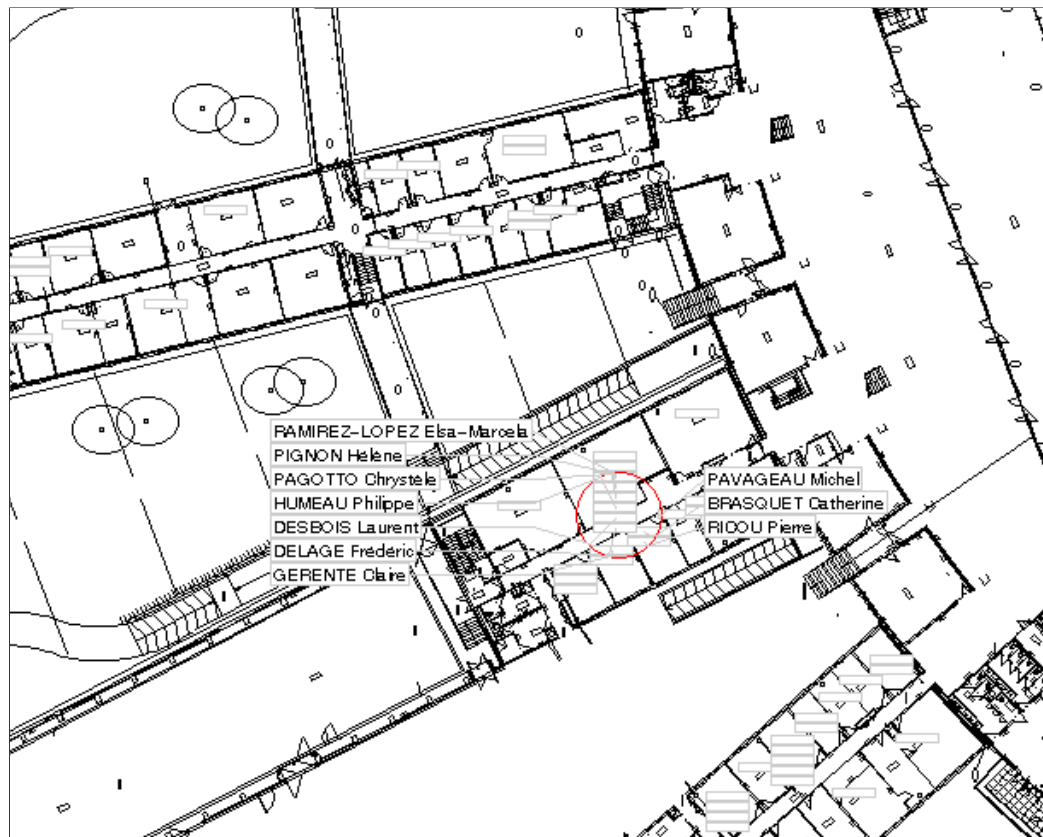


Figure 4.43: Larger Excentric Labelling example (from <http://www.cs.umd.edu/hcil/excentric/dist/bin/Eval.html>)

layout allows about 20 tightly grouped objects to be labelled in a readable way. The labels are sufficiently far from the cursor (the point that the user is most interested in) that the user's focus of attention remains visible. The labels are enclosed by differently coloured (the use of colour depends on the application) rectangles and these rectangles are connected to the appropriate objects with lines of the same colour. This allows the user to visually connect an object and its label.

Excentric Labelling is an effective way of adding temporary information to a display but complicated displays become even more overloaded with information (Figure 4.43).

4.3.9 Discussion

The semi-transparent objects discussed in this section are “layered” or depth multiplexed interface objects and allow users to maintain awareness of one task while concentrating on another. These objects address two user interface problems:

screen size and user attention. Transparent overview layers and the Macroscopic (subsection 4.4.6) economise screen space by showing two views of the information space at two different scales in the same window. Local tools such as Magic Lenses, Toolglasses, and Excentric Labelling allow users to display temporary transformations of the information space within the representation of this space. This both conserves screen space and removes the need for users to mentally combine two distant representations of the same information.

4.4 Zoomable User Interfaces

Zoomable User Interfaces (ZUIs) present an information space to users. Users change the scale of their view of the information space depending on the level of detail that they want to see at a given moment. As users zoom on an object it grows (optical zoom) until it vanishes and is replaced by other objects that represent the same underlying information but in more detail. This change in representation is called semantic zooming. Multiscale interfaces differ from focus+context views (section 4.1) in that only one scale is used at a time. There is thus no graphical distortion but objects that do not fit on the screen vanish and the context is lost. A further difference is that users have the impression of entering into the information space; their entire view of the space zooms as they move closer to the information of interest.

A Zoomable User Interface is based on the concept that data are organised in a two-dimensional virtual world. Users can travel in this world and focus on areas of interest (Furnas and Bederson, 1995). When users approach an object, the representation is modified and more details appear. They can also easily go to and return from semantically related objects. They can focus on a specific area or zoom out to have a global view of the data. This is achieved using the techniques detailed in this section. A more complete presentation of zooming, portals and lenses is given in Bederson et al. (1996).

4.4.1 Semantic Zooming

Furnas and Bederson (1995) discuss space-scale diagrams which can be used to visualize the 3D nature of objects in ZUIs and the changes in representation that result from semantic zooming. They represent both a 1D spatial world and its different magnifications explicitly in a single diagram. The objects visible at different magnifications, and their size at each magnification, are shown in the 2D representation of the 1D spatial world. An extra dimension, the scale, is added to the 1D world by piling up views of the 1D world at all the different scales (Figure 4.44). The user's viewing window is a slice through the space-scale diagram,

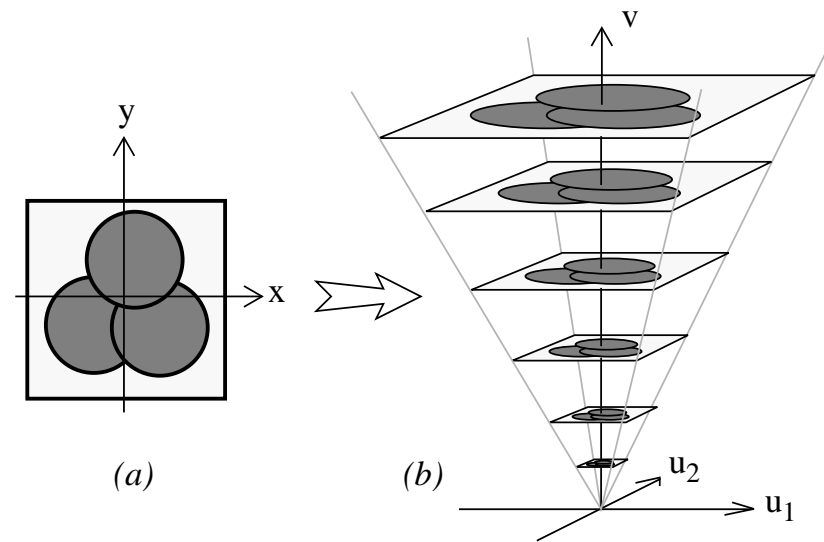


Figure 4.44: Construction of a space-scale diagram (adapted from Furnas and Bederson, 1995)

which when moved through the 2D diagram yields all the views of the 1D world. Figure 4.45 shows three movements in a space-scale diagram of a one dimensional space: (a) pan; (b) zoom; and (c) zoom on a point at the right hand edge of the window. These diagrams are used to study trajectories such as pan, zoom and joint pan-zoom. Joint pan-zooms are useful because in a multiscale world the shortest path between two points is not always a straight line. One way to move between two points is to zoom out until they are both visible then zoom back in. The fastest way to move between two points can be a more complicated path.

Zoomable User Interface space-scale diagrams can be used to visualize the transition points (when an object changes representation as a function of scale) and the nature of the changing representations. The slices at the bottom of Figure 4.46 show the objects visible at various different positions and scales. The view of the virtual world at Figure 4.46a does not include the dark grey object on the right hand side of the figure. When users zoom to view Figure 4.46b, they see part of this object and when they zoom to Figure 4.46c they see a second representation of this object. By the time they arrive at Figure 4.46d this object has vanished.

4.4.2 Special Objects in Zoomable User Interfaces

ZUIs can provide two different types of special objects. The first, portals, is fixed and created by the developer of the virtual world. The second, Magic Lenses, is defined by the developer of the virtual world and is created, moved and destroyed

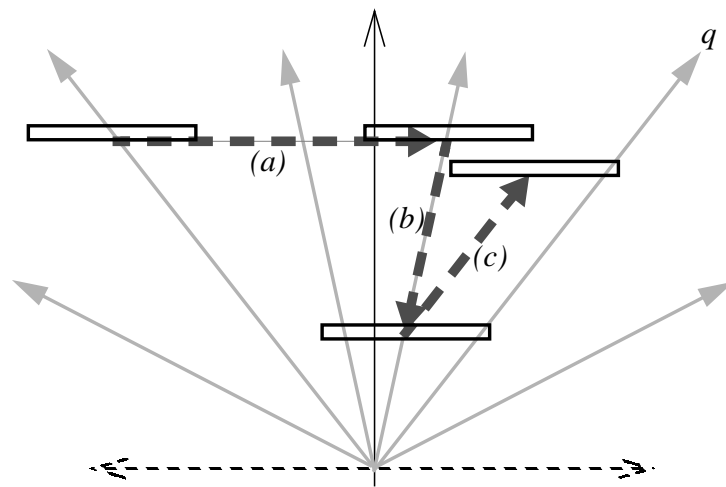


Figure 4.45: Basic trajectories in a space-scale diagram (adapted from Furnas and Bederson, 1995)

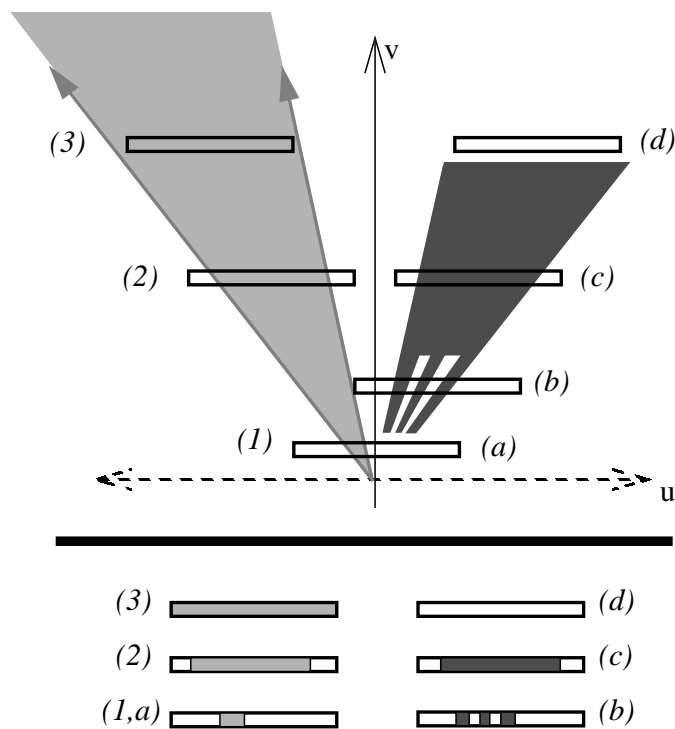


Figure 4.46: Semantic zooming in a space-scale diagram (adapted from Furnas and Bederson, 1995)

by users. Both these techniques have characteristics in common with multiple view interfaces. The difference is that these multiple views are integrated into the main view. A portal is part of the virtual world while a Magic Lens is an in-place user positioned transformation of the virtual world.

Portals

A portal is a special graphical object that gives a view of another part of the virtual world. It is generally static (as are those created with the tool presented in chapter 6); i.e. the user cannot create a new portal, only those provided by the system are available. It consists of a rectangle put at a given place in the world, in which another part of the world is displayed, often at a different scale. As with other objects, portals grow as the user zooms, and when a portal almost fills the user's view, the main view can be said to be transferred to the view in the portal. The user can manipulate (zoom, dezoom, or pan) the view seen through a portal simply by clicking inside the portal.

Portals can be used to express the semantic relationship between two objects that are widely separated in the virtual world, even though they are related. A portal can be placed near the first object, pointing to the second one. If necessary, a reciprocal portal can complete the symmetry.

Portals are useful to avoid the duplication of objects, a technique that is convenient in graph representation but complicates the understanding of the graph structure.

Magic Lenses

ZUIs use the Magic Lenses described in subsection 4.3.6 so that users can filter or transform the virtual world. These lenses can also be seen as portals that point to another part in the world where the representation is different. This adds a temporary fourth dimension to the two spatial dimensions and the scale "dimension".

4.4.3 Pad++

Pad++ (Bederson et al., 1996; Bederson and Hollan, 1995) is a zoomable graphical sketchpad. The computer screen is imagined to be made out of a new material that is stretchable like rubber but continues to show a crisp image no matter what the sheet's size. In addition objects are not just stretched geometrically; as they get bigger the representation of the object is adapted to the object's size. For example, when stretching a spreadsheet, the numbers do not keep on getting bigger and bigger, when there is enough space, the computations from which the numbers were derived are shown. The authors also implement Magic Lenses (subsection 4.3.6).

4.4.4 NaviQue

Furnas and Rauch (1998) present an information and navigation environment, called NaviQue, implemented using Pad++. They list four technologies that have become available recently: direct manipulation which allows the user to visualize and directly manipulate objects, information visualization tools (subsection 4.2.4 and subsection 4.1.2) for browsing large bodies of information, lenses (subsection 4.3.6), and infinitely zoomable (“multiscale”) interfaces.

NaviQue was designed with various desirable features in mind.

- The environment should integrate the broader information world and the user’s private workspace. This allows queries to be run on current work and the integration of search results into the initial task.
- All conceptual items should be given visual representations so that the user is aware of them and their statuses. This is called visual reification.
- The environment’s tools should be used by direct manipulation of visual representations of the tools.
- The tools in the environment should support the basic tasks of basic information seeking and analysis. When performing a query the user asks a search engine to look for a specified target in a specified collection. Browsing is performed by zooming and panning in a hierarchical structure of information. The results of queries and browsing are facilitated by allowing the user to create, copy group information objects on the workspace, and by showing the relationship of objects to their multiple contexts. The user can create new material such as private annotations and documents for distribution.
- NaviQue should provide multiple views where different objects can be seen or the same objects seen in different contexts.
- Furnas and Rauch (1998) state that users perform the same actions in two different ways: lightweight (cursory) and heavyweight (detailed). NaviQue should support these two types of interactions.
- The environment should support interactions with other users, ranging from asynchronous and informal “publication” of parts of documents, to tightly coupled synchronous conversations with co-workers allowing shared searches and authoring.

Objects in NaviQue are structured, fractal (are in a structure and have structure), queryable, navigable, historical, similarity engine compatible, contextualised, evalualised, annotatisable, updatable, and are able to contain metadata.

NaviQue was implemented using Pad++ and is thus a multiscale interface. All the information objects reside on the multiscale work surface; there is no separate query window (the user can create a text object on any part of the surface and use it as a query). Queries are performed using a vector-based similarity engine. NaviQue also defines electronic sets (*e-sets*). These allow the user to group objects, view them in different ways and perform actions on them. Each e-set has an associated widget. Moving the cursor over this widget highlights all the members of the set. The results of a query are returned in an e-set and are highlighted in two different colours in the corresponding query and collection e-sets, thus providing an in-context view of how the results relate to the query.

4.4.5 Information Density

Woodruff et al. (1998a) cite the *Principle of Constant Information Density* from the cartographic literature (Bertin, 1977; Tufte, 1998) which states that the number of objects per display unit should be constant. The amount of information should remain constant as the user pans and zooms. The information density can be maintained constant by showing objects at greater detail as the user zooms on them, by showing more objects as the user zooms, or both.

The authors extension to the multiscale interface, DataSplash, is aimed at the developers of multiscale user interfaces and allows them to interactively control when objects are visible as the user zooms. Figure 4.47 shows a map of the United States. At the current scale (indicated by the horizontal line in the upper right hand quadrant) only the state outlines and the dots indicating the positions of cities are visible. As indicated by a vertical bar for each layer, the state outlines are always visible but as the user zooms the dots will be replaced by circles indicating the cities size and position, and later their names will be added. The developer can move the vertical bars and immediately see the effect on the left hand side.

A later version of this program uses the width of the vertical bars to indicate the information density of each layer at each scale, and coloured tick marks to show if the information density at each scale is acceptable (Figure 4.48). The information density is just the number of objects visible.

There are eight variables that provide information in two-dimensional graphics: x coordinate, y coordinate, size, value, texture, colour, orientation and shape. DataSplash provides functions to modify the position, colour, size, and shape of objects in order to reduce or increase the information density. The user can thus modify the width of a vertical bar and the system modifies the objects in order to reduce or decrease the information density.

The program presented in Woodruff et al. (1998a) explored the information density in the scale (z) dimension only. A discussed extension is to consider whether the data is uniformly distributed in the x and y dimensions. They sug-

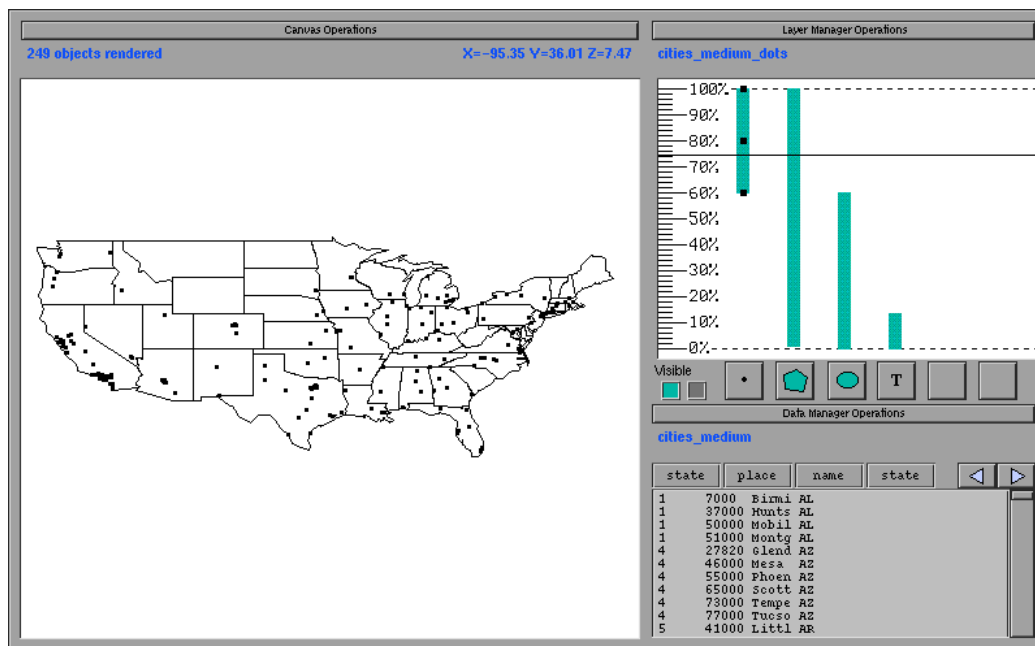


Figure 4.47: Displaying and controlling information density (adapted from Woodruff et al., 1998a)

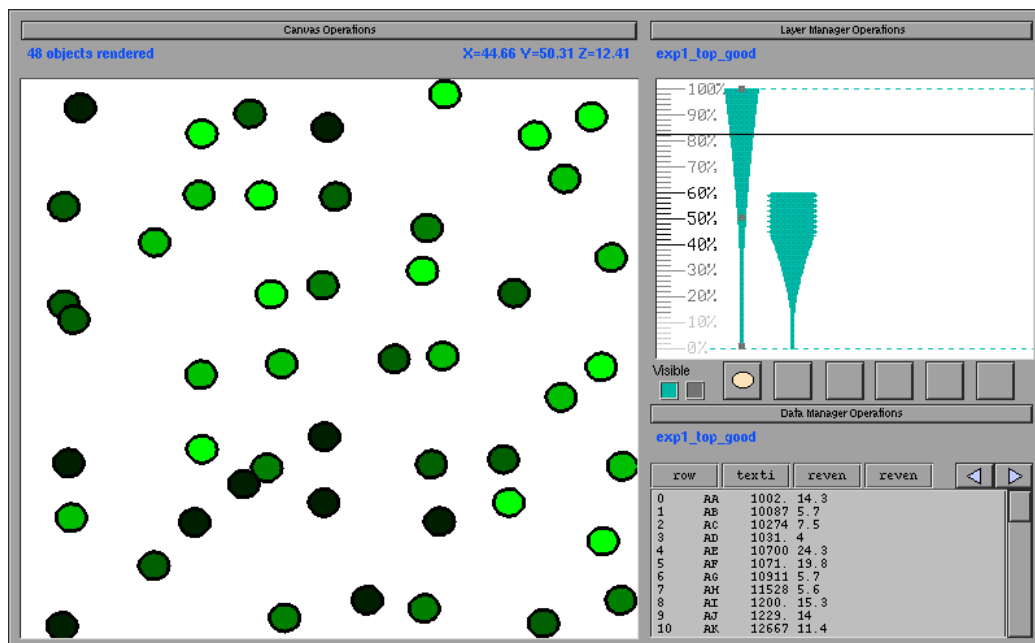


Figure 4.48: Display and control of information density (adapted from Woodruff et al., 1998a)

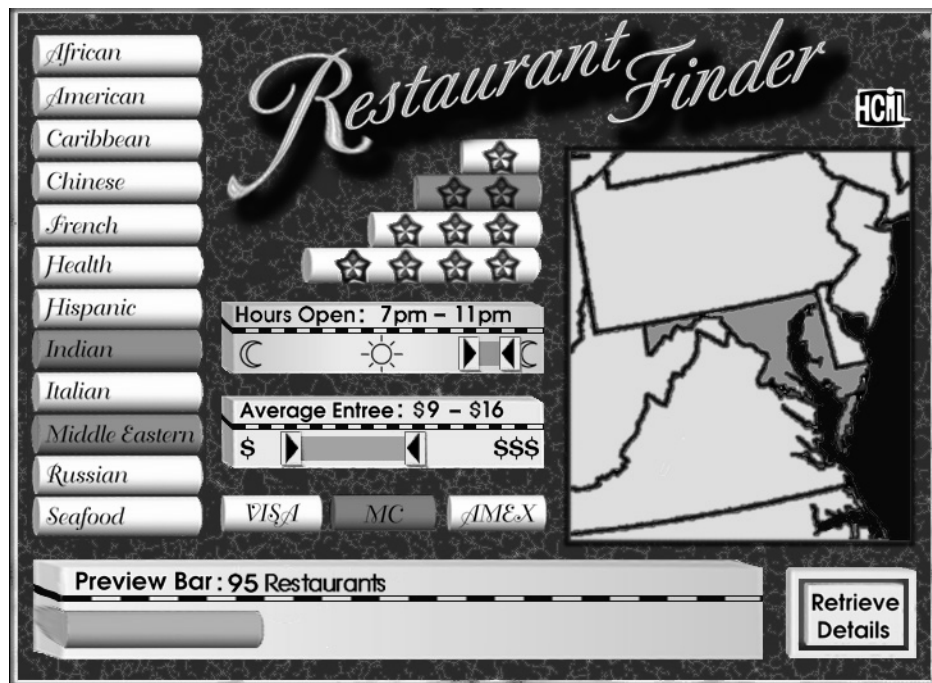


Figure 4.49: Restaurant Finder (adapted from Plaisant et al., 1999)

gest that each equally-sized subdivision of the information space should contain a constant amount of information. A program could show those regions that are too sparse or too cluttered.

This interface resembles the dynamic queries and query previews proposed by Plaisant et al. (1999). An example query preview interface is shown in Figure 4.49. The users have used the sliders to select only those restaurants open between 7pm and 11pm and where the main courses cost between \$9 and \$16. This differs from DataSplash in that it is designed as a database query tool for users rather than as a developer's aid for creating virtual worlds for ZUIs.

4.4.6 Macroscopic

Lieberman (1994, 1997) proposes a Zoomable User Interface, called Macroscopic, that uses multiple translucent layers. In a standard zooming user interface the user loses all context after a zoom. The Macroscopic maintains both the zoomed in and zoomed out views visible in overlaid translucent layers on the display.

The user zooms on a region by selecting a region, this region is then indicated on the still visible original image by a rectangle and the entire original image is overlaid by an enlarged view of the area in the rectangle. Moving the rectangle causes a pan and resizing the rectangle causes a zoom. The user can control the

translucency levels between the layers in order to emphasise the background to help in orientation or to emphasise the foreground in order to concentrate on the close-up view. Macroscopes can be extended to have three layers and, in this case and in general, moving one layer relative to the others helps users identify which objects are in which layer.

Figure 4.50 shows how a Macroscope version of the Macintosh Finder could be designed. Users do not open or close folders. They just zoom on parts of the disk to see the contents in more detail. Figure 4.50a shows a directory “brazil” and a subdirectory “Henry” as seen by a standard Finder. The Macroscope version (Figure 4.50b) replaces the opaque “Henry” window with a miniature view of its contents in the main window. If users want to see the contents of the “Henry” folder in more detail they zoom the region of the screen containing the folder. This is shown in Figure 4.50c. The original view is still visible but paler than before and the extent of region that was enlarged is shown drawn on this view (the box that surrounds the “Henry” folder cutting through the “System Folder” and the “Attic”). The enlarged region is drawn on top of the original view. Figure 4.50d shows the next step where the user has zoomed on the file containing Lisp code in the bottom right of Figure 4.50c. Once again a rectangle has been drawn around the enlarged region and this new region drawn on top of the previous view (which has now been faded out).

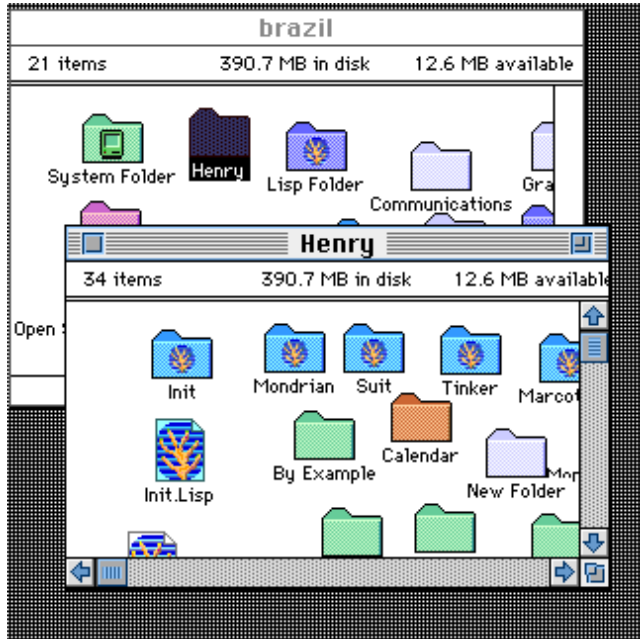
4.4.7 Goal-Directed Zoom

Woodruff et al. (1998b) state that in a normal multiscale interface the current elevation (or scale) determines the representation. They propose a different *goal-directed zoom* where the user specifies the desired representation of an object and the system zooms to the scale where the object is visible with that representation. In their system, users click on an object to get a menu of the possible graphical representations of the object. When the user selects a representation, the system pans so that the object is in the centre of the display and zooms so that the selected representation is visible.

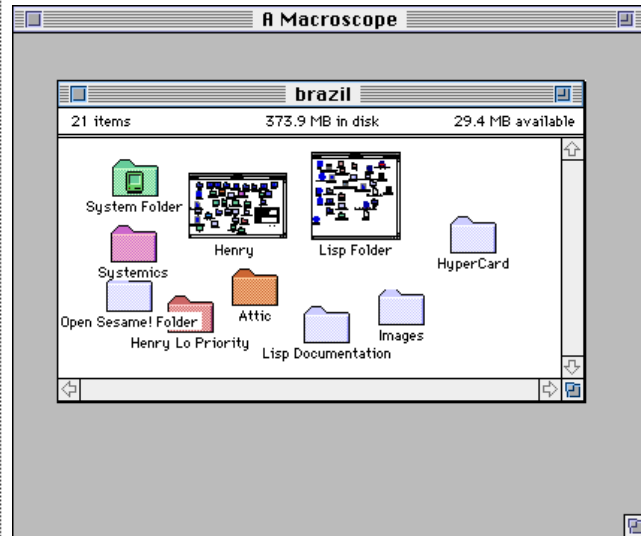
4.4.8 Desert Fog

Jul and Furnas (1998) introduce the notion of *desert fog*, regions in multiscale worlds which do not contain any navigational information. When users arrive in *desert fog* areas, they have no idea whether to zoom, dezoom, or pan to find an interesting view. Their solution to this problem requires the navigation system to modify the virtual world so that in a region that would otherwise be empty the user is shown what action should be taken to find a non-empty view. If a view contains objects that are currently invisible (because they are not visible at the

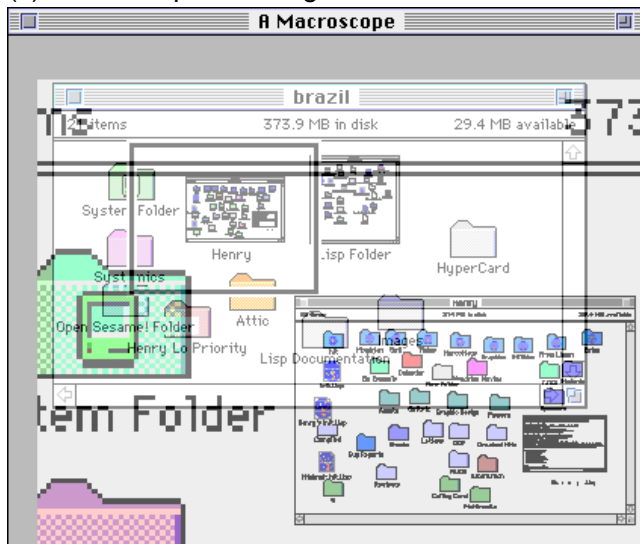
(a) two Macintosh folders in the Finder



(b) Macroscopic showing the top folder



(c) Macroscopic showing two folders



(d) two folders plus a zoom on a file

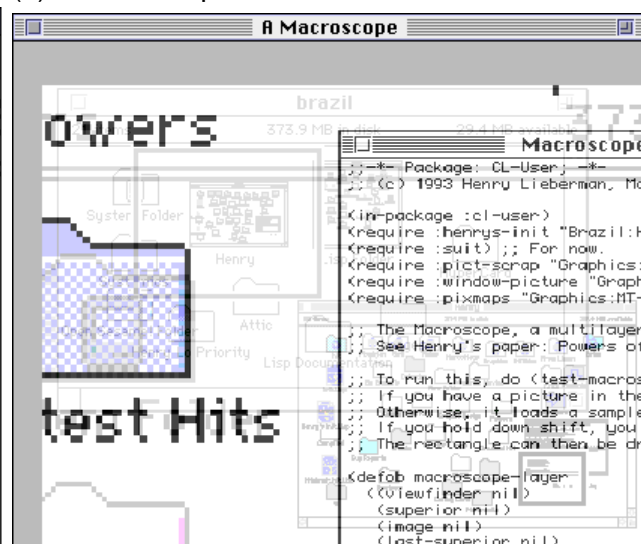


Figure 4.50: Macroscopic as a Finder (adapted from Lieberman, 1997)

current scale but will be visible after zooming) the system automatically generates *residue* which indicates where the user should zoom to see these objects. Thus if users are faced with an empty screen they know that they should just dezoom until they find objects. If users see *residue* they should zoom on the *residue* until the objects come into view. The problem is that when there are a large number of invisible objects there is too much residue. Another approach produces indicators from interesting views (those that contain objects), called *critical zones*, that are currently invisible but can be found by zooming. These indicators are multiscale and grow like normal objects except that they have minimum size in order to be always visible.

4.4.9 Discussion

The distorted views presented in section 4.1 have a variable distortion over the information space. Non-focal information is given less space (per unit of information) than focal information. Non-focal information is thus shown at a smaller scale than the focus. Zoomable User Interfaces have a constant scale at any moment. There is no space for non-focal information and it is not shown.

The method of deformation used in Zoomable User Interfaces is a simple form of pseudo-optic deformation (subsection 4.5.7) as graphics grow followed by logical deformation (semantic zooming) when the graphics are replaced by other graphics showing the same underlying information but in more detail. The logical deformation function depends on the type of information presented. This function chooses representation appropriate to the size of the region to be displayed and the screen space available.

Lack of Context

One of the reasons that users are sometimes unable to successfully use ZUIs is that the view of the information space shown to users, the focus, does not always contain the context needed by users to position this focus in the information space. This is true once users have navigated away from the initial global view and especially so once they have navigated long enough to no longer remember exactly where they are. Once in this situation users are disoriented, sometimes to the point of not understanding what they are looking at and not knowing whether they should pan, zoom, or dezoom to find what they are looking for. It could be said that they are “lost in hyperspace”.

The Macroscopic (subsection 4.4.6) proposes a solution to this disorientation problem. It always leaves the global (context) view visible and simply overlays it with the focal view. The position of the focus is indicated on the global view in

order to indicate the focus's position relative to the global view. One disadvantage of this solution is that screen becomes overloaded with context information even when the user does not need it. Another disadvantage is that the maximum difference in scale between the focus and the context is limited. If the difference in scale is too great then the global context will be too coarse, with respect to the focus and the user's current point of interest, to allow the focus to be accurately positioned.

Missing Structural Information

A second problem is a lack of structural information concerning the information space. The initial or global view of the information space shows users what objects are in the information space but does not contain an indication of what information concerning these objects will be shown when users zoom. Desert Fog (subsection 4.4.8) addresses a different problem. It fills in regions of the virtual world that are empty even though they contain objects that will become visible if users zoom on these regions.

4.5 Taxonomy

There are two important questions that need to be answered when comparing the visualization techniques described in this chapter: in what form is the information space presented to the users and how users control this representation.

4.5.1 Taxonomy of Distortion-Oriented Techniques

Leung and Apperley (1994) review possible solutions to the visualization problem: the small size of the user's screen and the large amount of data available. Their taxonomy, shown in Figure 4.51, classifies presentation techniques depending on whether or not the data to be presented is graphical and then whether distorted or non-distorted views are to be used. Some of the different visualization techniques presented can be classified using this taxonomy (Table 4.2). Other techniques such as Treemaps or three dimensional representations such as Cone Trees are not easy to classify using this taxonomy. A Treemap does not distort the data (except after zooming where some of the data has been removed). Three dimensional representations can be compared to distorted views because the view they show in two dimensions is always a distortion of the real three dimensional space in which the data has been projected. Three dimensional representations differ however from two dimensional distorted views in that users are

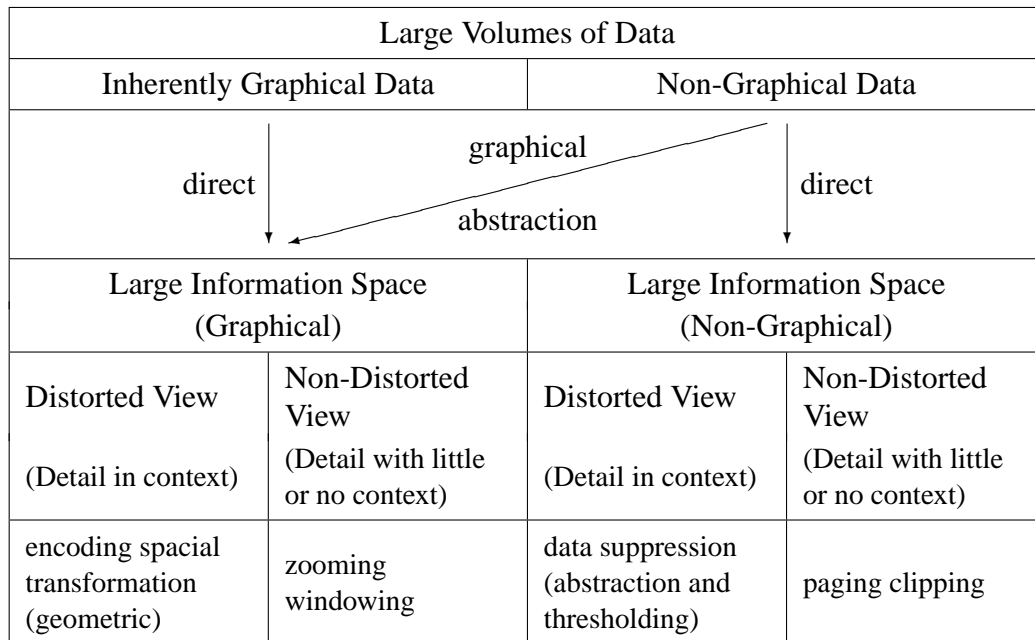


Figure 4.51: Taxonomy of presentation techniques for large graphical data spaces (adapted from Leung and Apperley, 1994)

Graphical		Non-Graphical	
Distorted	Non-Distorted	Distorted	Non-Distorted
Rubber Sheets 3D pliable surfaces non-linear transformations	Macroscope Zoomable User Interfaces Transparent Overview Layers Table Lens	fisheye views Perspective Wall	Macroscope Table Lens

Table 4.2: Classification following the taxonomy in Figure 4.51

meant to understand the three dimensional representation and remove the distortion to mentally create a non-distorted three dimensional image of the data. Three dimensional representations can also be said to make use of the third dimension to gain extra space to show the information space. Once this extra space has been used these three dimensional representations use user controlled pruning to reduce the amount of information to be displayed.

4.5.2 “Task by Data Type” Taxonomy

The “Task by Data Type” taxonomy proposed by Shneiderman (1996) proposes a “visual information seeking mantra”:

Overview first, zoom and filter, then details-on-demand

The method provides a task by data type taxonomy with seven data types and seven tasks.

Data Types

This taxonomy addresses the treatment of seven different data types:

1-Dimensional One Dimensional Data is linear data which includes textual documents, computer programs, and ordered lists. The original fisheye view (subsection 4.1.1) was used to visualize a computer program seen as a 1-dimensional information space. A computer program is not always considered as linear data. If the dynamic behaviour of the program is to be taken into account, computer programs are better viewed as networks (or directed graphs). The Document Lens (subsection 4.1.3) is another technique for visualizing a linear document. The document remains 1-dimensional even when it is laid out in a 2-dimensional grid as the vertical dimension has no meaning. The extra dimension was created just to reduce unused screen space.

2-Dimensional Two dimensional data includes maps, floor plans and newspaper layouts. Each item in the information space has a size and position in the 2D space. Rubber Sheets, 3D pliable surfaces, non-linear transformations, Transparent Overview Layers and Macroscopes are techniques for visualizing 2D data.

3-Dimensional This includes real world objects such as the human body, aeroplanes and buildings as well as models of these objects. The visualization techniques of this chapter do not treat these types of visualizations.

Temporal Time lines or temporal data are very similar to 1-dimensional data except that items have a start and finish time and may overlap. Many specialised tools exist but this type of data can also be visualized with a Perspective Wall.

Multi-Dimensional The information stored in relational databases is often best viewed as multi-dimensional data. Items with n attributes become points in n -dimensional space. The 2- or 3-dimensional scattergrams often used to visualize this sort of data are not otherwise discussed.

Tree The visualization of trees or hierarchies is discussed in section 4.2.

Network The visualization of networks or graphs (cyclic or acyclic, directed or undirected) is not discussed in this thesis but details can be found in Herman et al. (2000).

Tasks

After the description of the different types of data, this taxonomy defines the tasks that users might wish to accomplish with the data. Shneiderman states that these tasks are high level tasks. This is true in the sense that these tasks are abstractions of the exact actions that users are required to perform when using a data visualization program. These same tasks can however be seen as low level actions as they are what the users have to do to accomplish what they want to do. Users do not use information systems to “zoom” or “filter”. They are forced to “zoom” and “filter” to find the information or understanding that they are seeking.

Overview An overview, often a separate window from the view of the focus, helps users to gain an understanding of the entire information space.

Zoom Zooming allows users to increase the space allocated to more interesting information in the space.

Filter Users can filter the displayed data, often via dynamic queries, to remove uninteresting data items.

Details-on-Demand Once an data set has been reduced to a small number of items users can ask for more details.

Relate Relating data items is an extension of details-on-demand. An extra level of detail, the relationships between the objects, is shown when requested.

History A history mechanism is designed to support undo, replay and progressive refinement. It allows users to understand what they did to arrive in their current position.

Extract Extraction allows users to save either the end result of their queries (the resulting data items) or the queries themselves.

Comment

The taxonomy of Shneiderman is essentially static. The representation used to display the data is chosen depending on the type of the data and users are proposed tools that allow them to execute some (probably not all) of the listed tasks. Other than in the section on advanced filtering, little mention is made of the dynamics of the visualization systems nor how users interact with them.

4.5.3 Dynamics

The taxonomy of Chi and Riedl (1998); Chi (2000) takes a more dynamic approach to data visualization as it discusses the data transformations from the original format of the data to the format presented to users. It also includes in the taxonomy the operators that users can apply to the data at each transformation stage.

Figure 4.52 shows the Data State Model proposed by Chi. Column b in Figure 4.52 shows the four stages in the data visualization pipeline. The three operators, shown by column c, transform the data in each stage to the format in the following stage. In the example shown, the raw data is a web site. The web site is translated into a graph by reading the web pages and following the internal links. This operation is performed by the data transformation operator and gives the analytical abstraction. The visualization abstraction, a hierarchy, is then created by breadth first traversal (in the case of a Cone Tree, see subsection 4.2.5) by the visualization transformation operator. The users' view is created from the visualization abstraction by the visual mapping operator.

When using a disk tree, users can change from a breadth first traversal to a depth first transversal by changing the visualization transformation operator. The rest of the visualization pipeline remains unchanged.

At each stage of the Data State Model, users can modify the data (without modifying the data structure) by using the “within stage” operators. The operator at this stage, the value stage, is called the “value stage” operator. The web site is created and modified by an external tool that is not part of the visualization process shown in the example.

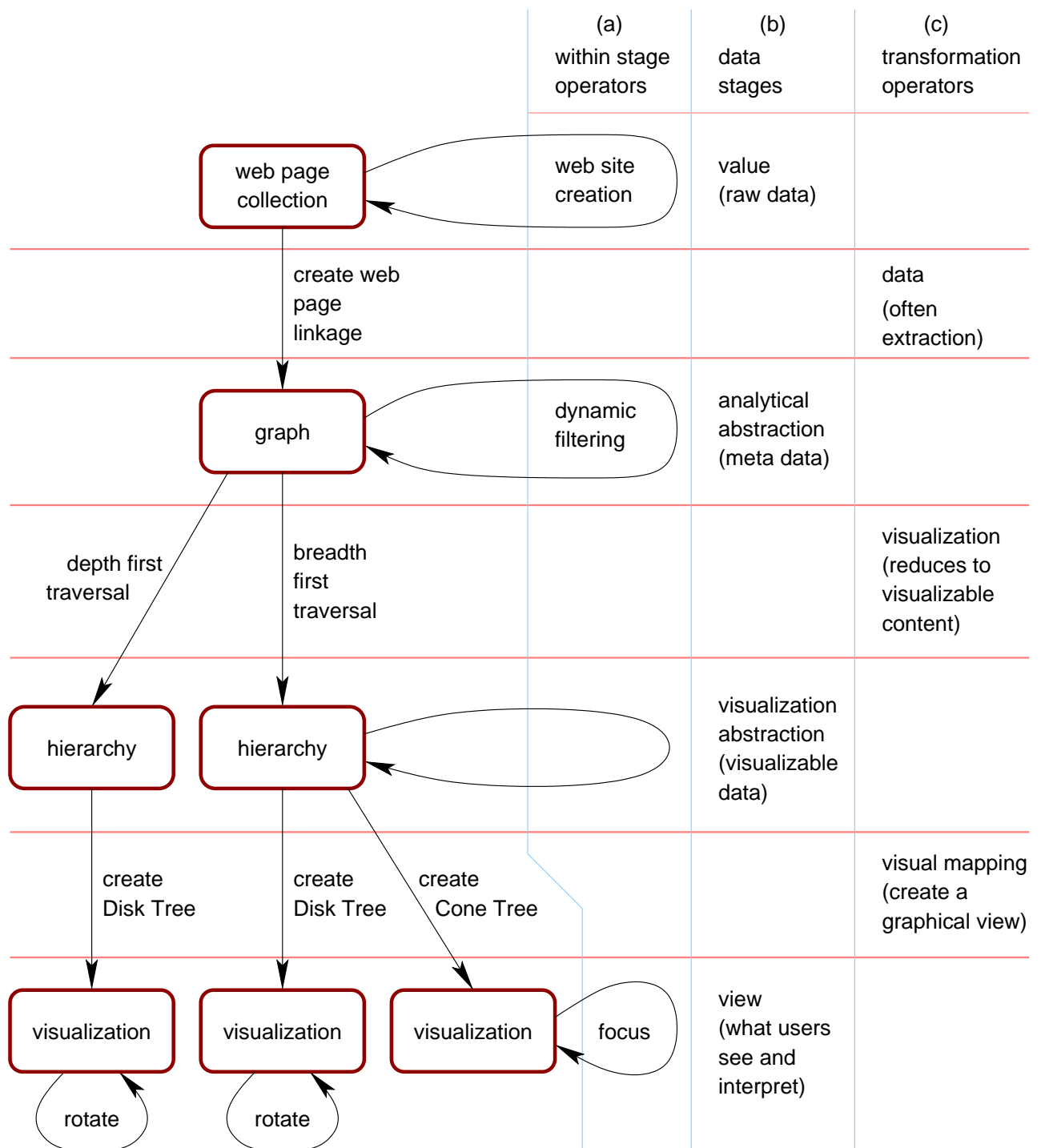


Figure 4.52: Data State Model applied to web sites (adapted from Chi, 2000)

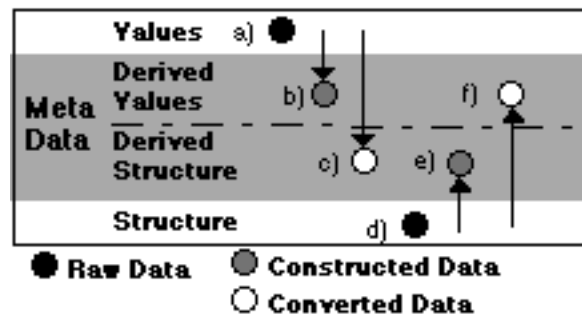


Figure 4.53: Types of represented information (adapted from Tweedie, 1997)

The data in the analytical abstraction is modified by the “analytical stage” operator. In the web site example the graph of web pages can be modified by dynamic value-filtering of nodes or edges. This filtering changes the nodes and edges used to create the Cone Tree. The data in the visualization abstraction can be modified by users using the “visualization stage” operator. The users’ view can be modified by the “view stage” operator. In the web page example and when a Cone Tree is used, the “view stage” operator allows users to change the focus node, rotate the tree, hide subtrees, change the orientation and position of the tree, and apply dynamic level filtering.

The Data State Model taxonomy allows visualization techniques to be split into the different parts of a pipeline and makes explicit where the various data transformations and users interactions occur. It does not however address the question of how users interact with a visualization system nor the important question of the relationship between interaction and visualization.

4.5.4 Interaction

Tweedie (1997) introduces the importance of interactivity in data visualization. It considers the data used to create representations, the interaction with users, and the input and output information that is visualized.

Data Representations

One part of the proposed design space comes from the different representations of data. There are data values (Figure 4.53a) and data structure (Figure 4.53d). Values are the numeric or categorical attributes associated with a problem while the structure includes the relations (for example links and constraints) that characterise the data as a whole. Different graphical techniques display different types of data. A histogram, for example, is well suited to displaying data values whereas

Do it Yourself ←————→ Command Mode				
Manual	Mechanised	Instructable	Steerable	Automated
Direct Manipulation				
		Indirect Manipulation		

Figure 4.54: Direct/indirect manipulation (adapted from Tweedie, 1997)

a tree diagram can represent relationships within a data set. Raw value or structural data can be transformed into meta-data. Data can remain in its original form as values derived from values (Figure 4.53b) and structure derived from structure (Figure 4.53e). This is constructed data. Data can also be converted data which comes from converting data into structure (Figure 4.53c) and structure into values (Figure 4.53f).

Interactivity

Most current visualization tools use direct manipulation. Direct manipulation was first defined by Shneiderman (1983). Tweedie (1997) defines direct manipulation interactions as literal replications of physical behaviour in the real world. They are direct as they involve manually moving an object or are mechanised via a tool metaphor. These two possibilities are shown on the left of Figure 4.54.

More recent visualization systems have started to make use of indirect manipulation. Tweedie defines indirect manipulation as interactors that cannot rely on direct physical metaphors because they provide behaviours that do not exist in the real world. “Magical” tools that stretch and deform objects in virtual reality environments are one example. These indirect manipulation interactors can be based on a tool metaphor or can be more intelligent as they move from being merely instructable to being automated (the right side of Figure 4.54).

Input/Output Relations Across Time

Successful use of indirect manipulation requires giving users visible feedback in response to their interactions with the visualization system. One way to provide feed back is to explicitly represent users’ input, tightly coupled to a representation of the results of that input (the output). Users can thus see the rules controlling the interaction and see and understand the effects of their actions. There are four types of relationship between input and output information: input → input (understanding two handed input), input → output (relating a scroll bar and the window’s contents), output → input (linking an error message with its cause), output → output (linking two output displays).

Users also need to understand the passage of time or how they have moved from one visualization to another. Linking the representations of past input and output to the state of the interface means that past input/output relations can be revisited. This allows users to understand how they arrived in a given state; it provides historical context.

Examples

Cone Trees With a Cone Tree (subsection 4.2.5) users see the structural information and can rotate the tree and move the required section into focus with manual direct manipulation. The input/output relation is limited to present: input \rightarrow output. No attempt is made to represent historical input/output relationships.

Pad++ Users of the Pad++ (subsection 4.4.3) Zoomable User Interface can visualize data and perform multi-scale zooming using mechanised indirect manipulation. Users can also filter the data using Magic Lenses, which is also mechanised indirect manipulation. The only input/output representation is input \rightarrow output.

Table Lens Users of a Table Lens (subsection 4.1.6) reorder the cells containing values using mechanised direct manipulation. Only output user interactions are represented.

Design Space

Tweedie (1997) identified five types of interaction: hiding/filtering, animated navigation, labelling/Boolean encoding, reordering, and algorithmic transformation.

Visualization programs can allow either direct or indirect interaction on either meta data or raw data. Indirect manipulation can be used to generate the meta data interactively and thus demonstrate the algorithm used.

Most visualization systems represent input \rightarrow output relations. A special case of this is the use of the “object symbol” (a Magic Lens for example) where both input and output are represented as a single entity. This proximity emphasises the relationship between them and provides visual feedback.

Output \rightarrow output relations are provided by systems that provide multiple representations of the same output.

4.5.5 Design Patterns

Vernier (2001) defines a number of design patterns that aid in the conception of

visualization techniques. We discuss three of these design patterns here: continuity spatial, continuity temporal, and understandable deformation.

Continuity Spatial

The continuity spatial pattern states that the focus should be integrated in the context in a way that assures the spatial continuity between the two. There are many examples of systems where this design pattern is respected. We have already presented the Perspective Wall (subsection 4.1.2), 3D Pliable Surfaces (subsection 4.1.5), and the combined linear and non-linear transformations described in subsection 4.1.8.

The advantages, listed by Vernier (2001), that come from following this pattern are:

- the display of the focus (magnified) and its context;
- the focus and its context are connected in such a way that moving between them does not require mental effort;
- the focus and its context are joined thus facilitating comparisons between the two;
- the focus is not deformed avoiding hindering work on the focus.

The last advantage is only realised if a combined linear/non-linear deformation is used. In this case the focus is effectively not deformed but the context is often severely deformed (for example in Figure 4.14). In addition, it is not only the context which is deformed, the transition between the context and the focus is also difficult to understand. Situating the focus in the context and making comparisons is thus made difficult.

Continuity Temporal

The continuity temporal pattern is applied when visualization programs change the view of the information space. This view can result from either a change in the representation of the same data, a change in the data being represented, or both together. The pattern is designed to:

- help users understand the changes in the view;
- stabilise the image in the view;
- analyse the movements made by navigation systems;

- help users follow the focus or find the new focus.

The solution proposed by the pattern is animation, that is to say a sequence of small movements in the view of the information space that helps users follow the changes in the view.

Understandable Deformation

The third design pattern states that any deformation used must be understood by users. Vernier asserts that:

- the continuity spatial pattern should be respected by using a non-linear transformation;
- users must understand that the information space is deformed;
- important text should not be made unreadable by deformation;
- users should be provided with visual aids to help them understand the deformation.

Discussion

The problem with the understandable deformation pattern presented by Vernier (2001) is that, as stated in subsection 4.1.8, non-linear transformations leads to views where part of the information space is severely distorted. Thus using a non-linear transformation to maintain spatial continuity often leads to views where the deformation is understandable but where the information around the focus is impossible to use. The deformations presented by Keahey and Robertson (Figure 4.14) allow users to understand the transformation of the view but at the price of making the information around the focus unusable. Users can no longer read the names of the metro station just out of the focus, nor read the text in the document that is just outside the focus. The same problem exists with 3D pliable surfaces (subsection 4.1.5). The non-focal information becomes so deformed that it becomes unusable.

The non-linear transformations presented in Keahey and Robertson and the transformations used in 3D pliable surfaces are independent of the contents of the information space. As these transformations are purely graphic and not take into account the semantics of the data in the information space they often produce unreadable representations of the information space. In contrast to these graphical transformations, the original fisheye views (subsection 4.1.1) are more usable because the deformations that they use are logical rather than graphic. The structure of a C program and the user's focus are used to decide what lines of the program

are important are thus to be shown. The chosen lines are drawn as standard text and are thus all readable.

Hyperbolic views (subsection 4.1.7) are also different from purely graphical transformations. Objects are positioned in the hyperbolic space and then, using the current position of the focus, those that fit are mapped onto 2D space. The representations, usually limited to circles and text, of these objects are not drawn deformed and are thus always readable. This is an example of a mathematical deformation of the information space.

4.5.6 Presentation Taxonomy

When an information presentation system needs to display two (or more) different parts or representations of the information space at the same moment the system can use space, time or depth multiplexing. When the screen is too small for the information that is to be displayed at a given moment then an information presentation system has to use either time or depth multiplexing. This section presents our taxonomy for classifying visualization systems according to how they multiplex their use of the screen.

Time Multiplexing

Time multiplexing means that the system presents different parts or representations of the information space at different moments. What is being presented at any moment can be controlled either by the system or directly by users. The principal disadvantage of time multiplexing is that users have to remember what was displayed so as to be able to relate it to what will be displayed next.

Deformation based tools are in part time based as at a given moment some parts of the information space might be given so little screen space that they are invisible. In addition, other parts of the information space are visible but, if non-focal, probably not readable. Users have to move these regions into the focal region if they want to read them.

Depth Multiplexing

Depth multiplexing means that different parts or views of the information space are being presented at the same time in the same screen space. Users have to use depth clues to understand which on-screen graphic belongs to which layer. The use of a depth strategy often leads to an overloaded display and sometimes requires considerable effort on the behalf of users in distinguishing the different layers.

Depth multiplexing is used by Macroscopes, Transparent Overview Layers, and the transparent tools presented in section 4.3. Three dimensional tools such as Cone Trees also use depth multiplexing in some sense as users are given two dimensional clues that must be used to create a three dimensional space.

Space Multiplexing

Space multiplexing means that the two different views are shown at the same time but in different screen regions. Two of the principal disadvantages of space multiplexing are the screen real-estate used and the time users have to spend shifting their attention from one area of the screen to the other.

The gIBIS system (subsection 4.2.1) uses space multiplexing to display two representations of the information space in parallel. Many other common programs (for example Microsoft Office) use space multiplexing to display two different views of a document at the same time.

4.5.7 Deformation Taxonomy

The sample of distortion oriented visualization systems presented in this section all have the common characteristic that they attempt to show the entire information space at all times. The screen space available to a visualization program is (almost) always smaller than the information space to be shown. These systems thus use deformation to make the representation of the information space fit in the available screen area. This deformation is based on a degree of interest function. Every piece of information is given a degree of interest that depends on the goals of the users and information's type. Information that is classified as being less interesting is given less screen space than more interesting information. This section describes our taxonomy that can be used to classify visualization systems according to how they deform the information space.

Logical Deformation

The first type of distortion technique uses an understanding of the underlying structure of the information space. If a region of the information space does not receive enough screen space to allow it to be displayed in its entirety then an application specific (that is, dependent on the type of information being visualized) function is used to create a summary, that fits in the available space, of the region. This summary is then displayed non-distorted. This type of deformation was called a "logical focus+context view" by Herman et al. (2000).

The fisheye viewer of C program text (subsection 4.1.1) is one example of this technique. Those parts of the C program that are uninteresting (i.e. distant from

the user's current line in the program) are given very little screen space. They are summarised by a function that understands C programs. This function will convert, for example, the control block surrounding, but a little distance from, the current line into just the line that contains the `for` or `while` that controls the block. If there is space the lines that declare the variables in the block might also be displayed. The chosen program lines are shown as plain non-deformed text.

Another example is the Table Lens (subsection 4.1.6). Here users decide which rows and columns are important. These rows and columns are given space enough to show the numerical value contained in the cells. Other, non-focal and thus less interesting, cells are only given enough space to show a histogram of their values.

The advantage of this method is that the deformation is adapted and optimised for the type of data being displayed. This method also avoids the pseudo-optic deformation that is often time consuming to calculate and frequently leads to unreadable text or graphics.

The disadvantage of this type of deformation is that it requires a deformation function that understands the information space. A new function thus has to be developed for each type of information space.

Position Deformation

The second type of deformation uses a function to change the position, and to a lesser extent the size, of objects in the information space. This function is designed to leave the interesting (focal) objects at the centre of the user's display while pushing non-focal objects to the edge of the display. The objects are positioned such that each focal object is sufficiently far from other objects so that focal objects can be labelled without the labels overlapping. The objects' labels are not deformed and thus remain readable. This and the following deformation methods are called "geometric" by Herman et al. (2000).

The hyperbolic display (subsection 4.1.7) uses a complex function to position objects on the screen. This function takes into account the user's current focus and the space required to label objects (which itself depends on the user's focus). The objects are then drawn and labelled with non-deformed graphics.

Rubber Sheets (subsection 4.1.4) are somewhat similar. A complex function is used to position the objects and then they are labelled with non-deformed text. Rubber Sheets differ from hyperbolic displays in that in some regions of the display the box used to represent the object is elongated or deformed.

The advantage of this method is that it requires a less complete understanding of the information space than logical deformation. The function that positions the objects only has to solve a positioning problem. Logical deformation requires an understanding of the meaning of the information. Position deformation does not.

It only requires an understanding of information's structure.

The disadvantage of this method is that the spatial relationships between objects are often destroyed by the deformation. This problem is exacerbated by the lack of information given to users that would allow them to understand these deformations.

Pseudo-Optic Deformation

The third type of deformation, pseudo-optic deformation, does not remove information from the display nor reposition objects in the information space nor separate the deformation of objects positions from their appearance. They simply create an image of the information space (much too big to fit on the screen) and then deform it in such a way that it fits on the user's screen and that the focus of the user's attention remains readable.

Perspective Walls (subsection 4.1.2) and Document Lenses (subsection 4.1.3) are simple examples of this method. A one or two dimensional image of the information space is created. Part of this information space is then bent away from the users. The text or graphics in this area is thus deformed and becomes difficult or impossible to read.

3D pliable surfaces (subsection 4.1.5) and non-linear transformations (subsection 4.1.8) are more complicated examples of this deformation method. Here the deformation is more pronounced and parts of the graphical image unreadable.

The advantage of this method is that the deformation function does not need to understand the information space. It is also independent of the graphical layout of the information. In addition, the entire image is deformed, which makes it easier for users to understand the deformation.

One disadvantage of this method is that the deformation leads to text and graphics becoming unreadable. The Document Lens for example devotes a large part of the screen space to showing unreadable text. This text will be of little use in providing context to users.

Summary

The distortion-oriented presentation techniques presented in this section can be classified into three different types of distortion. This classification is shown in Table 4.3. Logical distortion is the more difficult to implement but ensures that only readable text and recognisable graphics are shown.

Logical Deformation	Position Deformation	Pseudo-Optic Deformation
fisheye view	hyperbolic display	Perspective Wall
Table Lens	Rubber Sheet	Document Lens
		3D pliable surfaces
		non-linear transformations

Table 4.3: Distortion methods taxonomy

interface type	multiplex	deformation
pan and zoom	time	optical
multiple windows	space	none
deformation based tools	time & space	any
transparent tools	depth	position
semantic zoom	time	optical & logical

Table 4.4: Visualization taxonomy summary

4.6 Conclusion

The visualization techniques presented in this chapter can be divided into five different interface types and these different types of interface each use one of the three methods of presenting information spaces too big to fit on users' screens. A summary of these techniques and associated strategies is given in Table 4.4. The techniques that use semantic zooming, Zoomable User Interfaces, use time multiplexing of the available screen real-estate. We will see in the next chapter how it is possible to combine space, time and depth multiplexing to create more powerful ZUIs. These new ZUIs provide the context that is generally missing in Zoomable User Interfaces.

Chapter 5

New Context Aids for Zoomable User Interfaces

Zoomable User Interfaces (ZUIs) suffer from an important problem: user disorientation. The information space or virtual world presented in these interfaces is often very large and users have to move or zoom through many different graphical representations of the data before finding the desired information. This navigation in the information space can cause users to become lost. When in this situation users no longer know where they are in the information space nor the meaning of the objects currently visible.

As discussed in the previous chapter Zoomable User Interfaces are currently time multiplexed interfaces. In this chapter we will investigate the use of space and depth strategies in ZUIs. The extra visualization “room” that space and depth multiplexing of the user interface gives us will allow us to include new context and navigation aids in our ZUI.

We also make use of the fluid and continuous control provided by our new interactor, our Control Menu, to increase the ease of navigating in ZUIs. With this improved control we are also able to investigate manipulation techniques where the dynamic nature of the interaction and resulting display is just as important as the static form of the interface.

This chapter presents three navigation aids designed to reduce user disorientation. The first, hierarchy trees, is designed to help users to better understand the information space, their position in the space, and what information is available. The second, the context layer, helps users reorientate themselves if they become lost. The third, the history layer, shows users the path taken through the information space to arrive at their current position.

5.1 Hierarchy Trees

Hierarchy Trees were created to make the implicit hierarchy in ZUIs visible to users so that they can use it as an orientation and navigation aid. This aid is a second window in the ZUI and shows at all times the structure of the information and the users' current position within this structure. Users can also use this window to change their position in the information space.

5.1.1 Zoomable User Interfaces Visualize Hierarchical Information Spaces

ZUIs are often used on hierarchically structured datasets with a known structure. Creating an information space for a ZUI requires the developer to provide graphical objects visible in the top level view of the space that summarise those objects found when users zoom. These objects will then summarise those objects to be found as users continue to zoom. A hierarchical structure is thus created. Objects in the information space that do not follow this hierarchical structure, i.e. that are not accessible via objects visible in the top level view, will be hard to find by users because they will have no way of knowing where to zoom to find them.

ZUIs can also be seen as three dimensional spaces (subsection 4.4.1) where the scale is the vertical dimension. The main view in a ZUI shows a horizontal slice through the information space. We propose a second orthogonal view, called a *hierarchy tree*, that is a flattened vertical slice through the information space. It is not possible to show all the objects in the information space in this view. The developer of the information space must thus associate a type, a simple textual string, with all the objects in the information. It is this type of information that is shown in the hierarchy tree. The hierarchy implicit in the design of the information space is made explicit by the developer. When the developer positions an object in the information space, a parent object and a type is associated with the object. Using this information the ZUI can create a summarised view of the types of information available for the users. This view is in the form of a tree: the hierarchy tree.

5.1.2 Make the Hierarchy Visible

The hierarchy tree is shown as soon as the ZUI starts in an separate area on the right hand side of the interface (Figure 5.1). (It is thus a space multiplexed strategy.) Users can see what types of data are available in the information space and how they are organised. In Figure 5.1 the users can see that this information space consists of some marker indexes and a number of chromosomes, that these

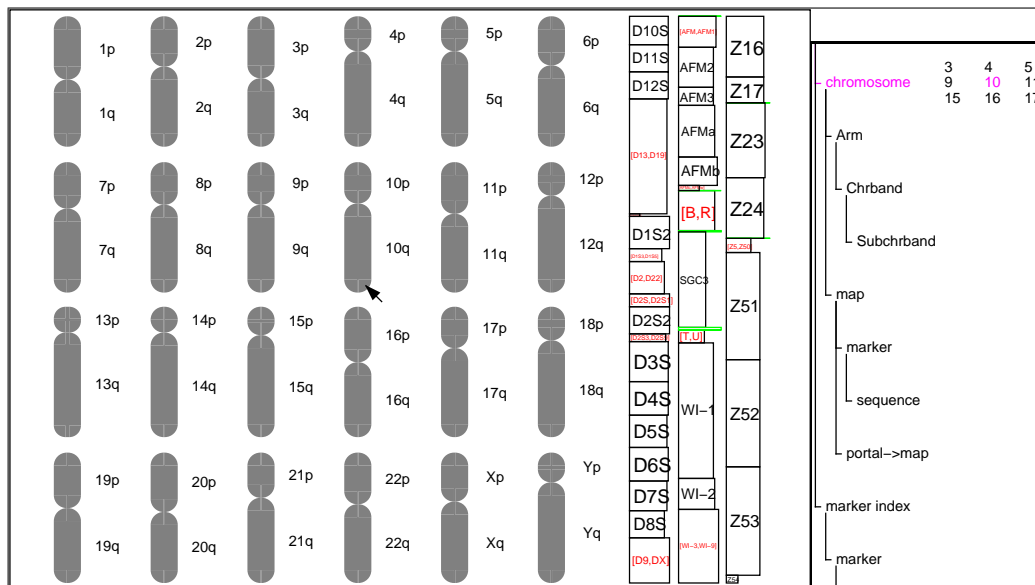


Figure 5.1: Toplevel hierarchy tree with the cursor on an object of type “chromosome” and name “10”

chromosomes contain arms and maps, and that maps contain the markers and the associated sequences. With this overview, users know that the information space contains sequences and that they need to zoom on the maps to find them.

If the hierarchy is too big to fit into the height of the window, then a sub-tree of the hierarchy can be opened and closed as in many standard file system browsers. One of the other techniques (described in section 4.2) could also be used if the hierarchy is exceptionally large.

5.1.3 Make the User’s Position in the Hierarchy Visible

As users move the cursor across the information space, the hierarchy tree is modified to show the type of the object under the cursor. This is done by changing the colour of the name of the type associated with the object and all those above it in the hierarchy of type names. When the developer creates an object the name of the object is supplied along with its type. The names of the object under the cursor and all the objects above this object in the information space are shown in the hierarchy tree. In Figure 5.1, the cursor is currently on the chromosome number 10. The text “chromosome” and the number “10” are thus shown in magenta (or gray) in Figure 5.1.

When users zoom on a particular type of information, they will soon see a view of the information space where all the visible objects are children of a given

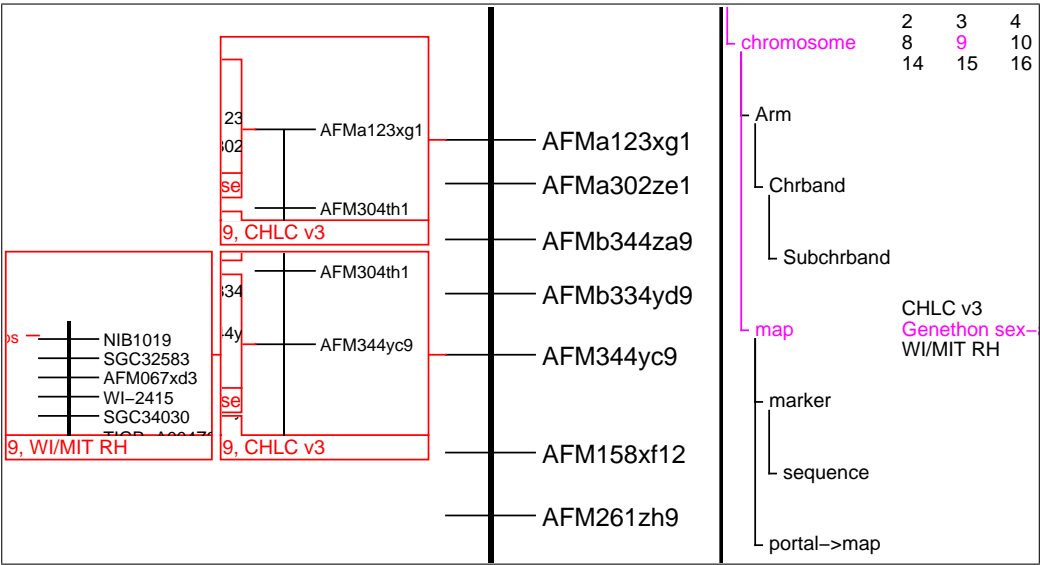


Figure 5.2: Hierarchy tree showing part of the “Généthon” map on chromosome 9

type. When the cursor is not in the interface or when cursor is not on an object, the hierarchy tree shows the lowest level in the hierarchical structure of the information space that is the parent of all the visible objects. Figure 5.2 shows the hierarchy tree after the users have zoomed to a point where the all the currently visible objects are part of the “Généthon” map on chromosome 9. In this situation, the name of the “Généthon” map, the name of the associated type, “map”, the number 9, and the name of the associated type “chromosome” are shown in magenta (gray).

At each level in the tree that includes the current object or objects, the names of those objects around the current object are also shown (Figure 5.2).

5.1.4 Using the Hierarchy Tree to Navigate

This technique offers an efficient method for rapid movement in the virtual world as users can use the hierarchy view to navigate in the information space and to directly access related but currently invisible objects.

When all the visible objects are descended from another object, the type and name of this object and its ancestor objects are shown highlighted on the display. Users can click on these strings to “move up” to a less detailed view in the information space. The ZUI dezooms the view of the virtual world until the rectangle that contains all of the objects descended from this object is wholly visible. This allows users to move up in the information space to see the representation of a part of an object in the information space in its entirety.

It would be possible to use the hierarchy tree to zoom to see a more detailed view of the information space. The user could click on the name of a type below the current scale and the ZUI would zoom to show an object of that type. It would not however be possible for users to choose the object to be shown among the possibly very large number available. The ZUI would have to choose an object and in most cases this choice would have to be random. We have not investigated this possibility any further.

If the user clicks on the string “chromosome” shown in Figure 5.2, the ZUI will dezoom sufficiently to show all of chromosome 9. A user could also click on the string “Généthon” to dezoom the interface so as to see all of the map “Généthon” on the chromosome 9.

The names of objects surrounding the current object and of the same type are also indicated in the hierarchy tree. The strings can also be used to navigate in the virtual world. When users click on the name of an object, the interface moves to show this other object. The current scale is not changed and the users are positioned at the same relative position on the new object. A user can select the string “CHLC v3” (the name of the map next to the “Généthon” map on the chromosome 9) to move to map “CHLC v3”. In this case, the ZUI will show the same relative position on the new map while maintaining the same scale. The new view will be at the same distance from the start of the map “CHLC v3” as the old view was from the start of the map “Généthon”.

5.1.5 Comparison With Other Techniques

This section compares Hierarchy Trees to some existing techniques and presents some of the advantages of this new technique.

Tool Tips

Users do not have to ask for the information provided by the hierarchy tree. In this respect it is similar to the “tool tips” that are found in many common computer applications and the Excentric Labelling technique (subsection 4.3.8). The advantage of the hierarchy tree is that the information is more complete and the information does not clutter the main view (directly under the user’s focus of attention). The hierarchy tree can also be used for navigation which is not the case for “tool tips”. The price paid for this additional information is the space used by the hierarchy tree and the fact that users have to divide their attention between two physically separate views. Users have to mentally combine these two separate views into a single composite understanding of the information.

Critical Zones

The critical zones technique (subsection 4.4.8) can be used to automatically provide the clues (additional objects in the virtual world) that there are objects to be found by further zooming. Hierarchy trees are different in that the developer of the virtual world is assumed not to create objects that are not accessible by zooming on objects accessible from the top level view. Hierarchy trees are designed to indicate that information is visible and on what objects the users should zoom to find this information.

gIBIS

The gIBIS system (subsection 4.2.1) provides a node index of the displayed IBIS graph structure that is in some ways similar to our hierarchy trees. This node index or global view shows the subject of all the nodes in the network organised by their primary link. As users zoom or pan the local view of the network, the global view scrolls to show users their current position in the network. The global view can also be used to navigate in the network. This view does however differ from our hierarchy trees in that it shows all the nodes in the network. This means that at any moment only a small proportion of its contents are visible and a scroll bar must be used to navigate within the global view. ZUIs typically contain a very large number of objects and so a global view of all the objects in the information space would be so big as to be ineffective. As discussed above, our hierarchy trees are designed to make use of the structure present in many ZUIs and thus show the types of the objects in the information space and the names of only that object under the cursor and its ancestors in the hierarchy. The hierarchy tree is much smaller than the list of all the nodes in the information space. It is thus easier to assimilate.

5.1.6 Limitations in Open Information Spaces

It is not possible to create a complete hierarchy tree for a virtual world that is open. If there is no upper or lower limit on the scale in the virtual world then the hierarchy tree can only show the types of information around the user's current scale. This situation would lead to a hierarchy tree that is extended as users zoom and dezoom.

A further restriction is that the hierarchy tree must be able to be calculated quickly. A virtual world used to browse the World Wide Web, where collecting exhaustive information is effectively impossible, would have to limit the time taken to generate the hierarchy tree. The hierarchy tree would thus have to be incomplete but could be extended as a background task as users navigate. The

direction of the users' navigation would guide the background task as it seeks to extend the hierarchy tree.

5.2 Context Layer

In ZUIs users can only see one view at a time: the focus. Users often cannot understand where the focus fits into the information space because it only shows a limited region of this space and ignores the surrounding context. The view of the focus shown in Figure 5.3d shows a view that the user might see after having navigated for a while in the ZUI. This view does not contain any clues that would let the user know what map or chromosome is visible.

Our new aid, the *context layer*, is displayed by user request command if they become disoriented and allows the users to position the focus with respect to more global views of the information space.

5.2.1 Context and Focus Both Visible

In contrast to the methods described in section 4.1 that integrate the context and focus by deformation, we integrate the context and the focus by temporarily drawing the context (the context layer) over the focus. There are thus two views visible, overlaid, on the user's screen: the context and the focus. This is depth multiplexing of two views into a single space. Figure 5.4 shows how the context layer is constructed. The users have arrived in the situation shown in Figure 5.3d and are lost. They then ask for the context layer which gives Figure 5.4b. This view consists of the context, in this case the top level (Figure 5.3a), drawn over the focus. The position and size of the focus relative to the context is indicated by a rectangle drawn on the context. This rectangle is indicated by the arrow in Figure 5.4.

5.2.2 Context Layer is Temporary

Our ZUI normally only shows the focus (and the hierarchy tree discussed in section 5.1). Users can create (and control) the context layer with a single gesture. Once this gesture is ended (by releasing the mouse button used to start the gesture) the context layer vanishes and the users return to the view they had of the virtual world before the creation of the layer. The fact that the context layer is temporary avoids surcharging the user's view of the information space with context information that is often unnecessary. The context layer can however be created with a simple mouse movement that makes its creation when necessary as simple and rapid as possible.

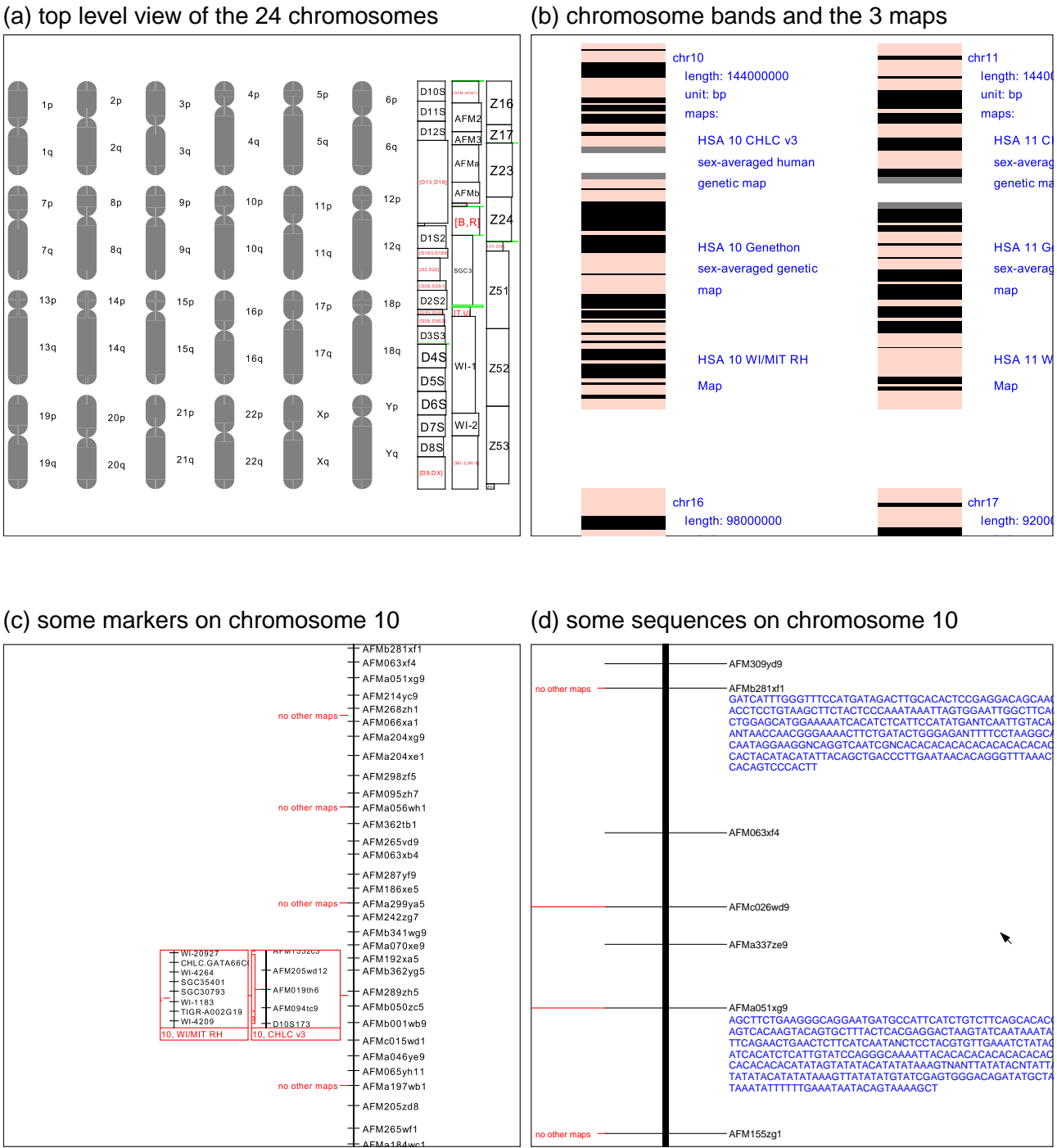
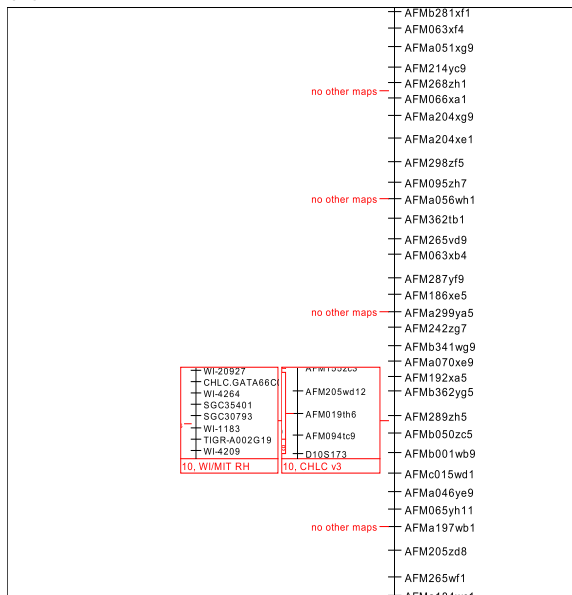


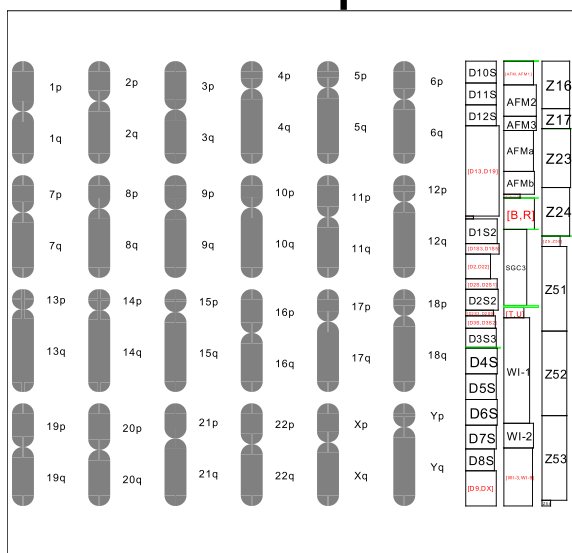
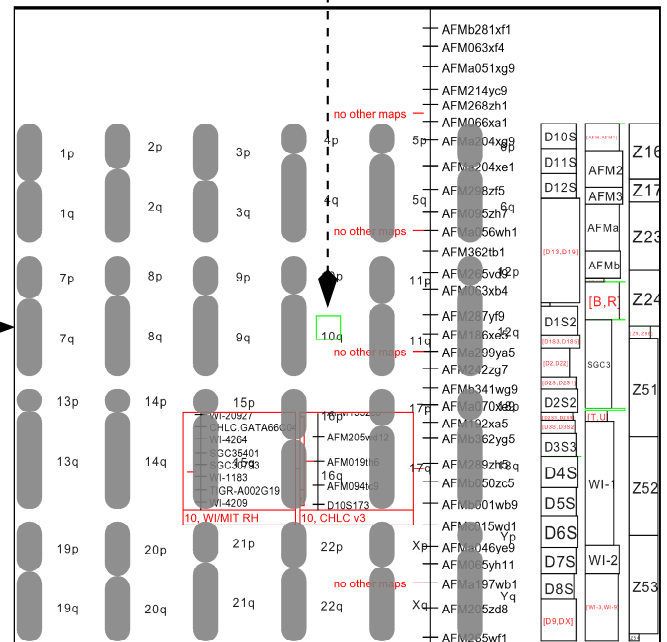
Figure 5.3: Lost after zooming

(a) focus



position and size
of the focus
relative
to the context

(b) context layer



(c) context

Figure 5.4: Constructing the context layer

5.2.3 Transparency Used

When the context layer is being used two views are visible on the user's screen. One of these views is transparent so that they can both be seen at the same time. Users are able to control which of the two views is transparent and its level of transparency. The users can decide which view is the more important at a given instant, make that view solid and fade out the other view as much as necessary. The control of the transparency must be as fluid as possible so that the user can move very quickly from concentrating on one view to concentrating on the other. This fluidity is obtained by the use of a Control Menu (explained later in this chapter).

Figure 5.5b and Figure 5.5c show the same focus and context. In Figure 5.5b the focus is drawn solid and the context is transparent. In Figure 5.5c, the users have switched their attention to the context. It is drawn solid while the focus has been drawn transparent. The user studies discussed in section 4.3 (Harrison et al., 1995b) indicate that transparent views and overlays are well accepted by users.

Systems that show the focus and the context in two different display areas are space multiplexed. Systems that show either the focus or the context (at the users demand) in the same display area are time multiplexed. Context layers are depth multiplexed (subsection 4.3.7). They are shown at the same time in the same place and the users use their depth perception, aided by the interaction, to separate the focus and the context. They can also be said to be space multiplexed because context layers are only visible when required by the users.

5.2.4 Context is Shown at Many Different Scales

The global view visible in the context layer is not fixed. The users can choose and rapidly change the scale of the global view. It is however always centred around the focus. This allows the users to start with a global view that is similar to the focus and then rapidly change the scale of the context layer (dezoom this layer). This continuous choice of contextual views allows users to position the focus in the many different contexts and to thus understand the location in the information space.

Figure 5.5a shows the top level or initial view of the information space drawn over the focus. The rectangle (indicated by the arrow) that indicates that size and position of the focus is small as the focus is small compared to the global view. The rectangle in Figure 5.5a is drawn over the chromosome 10. This indicates that the focus is showing part of this chromosome.

Figure 5.5b shows the same focus as in Figure 5.5a because the focus never moves during the use of the context layer. The context in Figure 5.5b has however been zoomed. The rectangle that shows the size of the focus has grown because

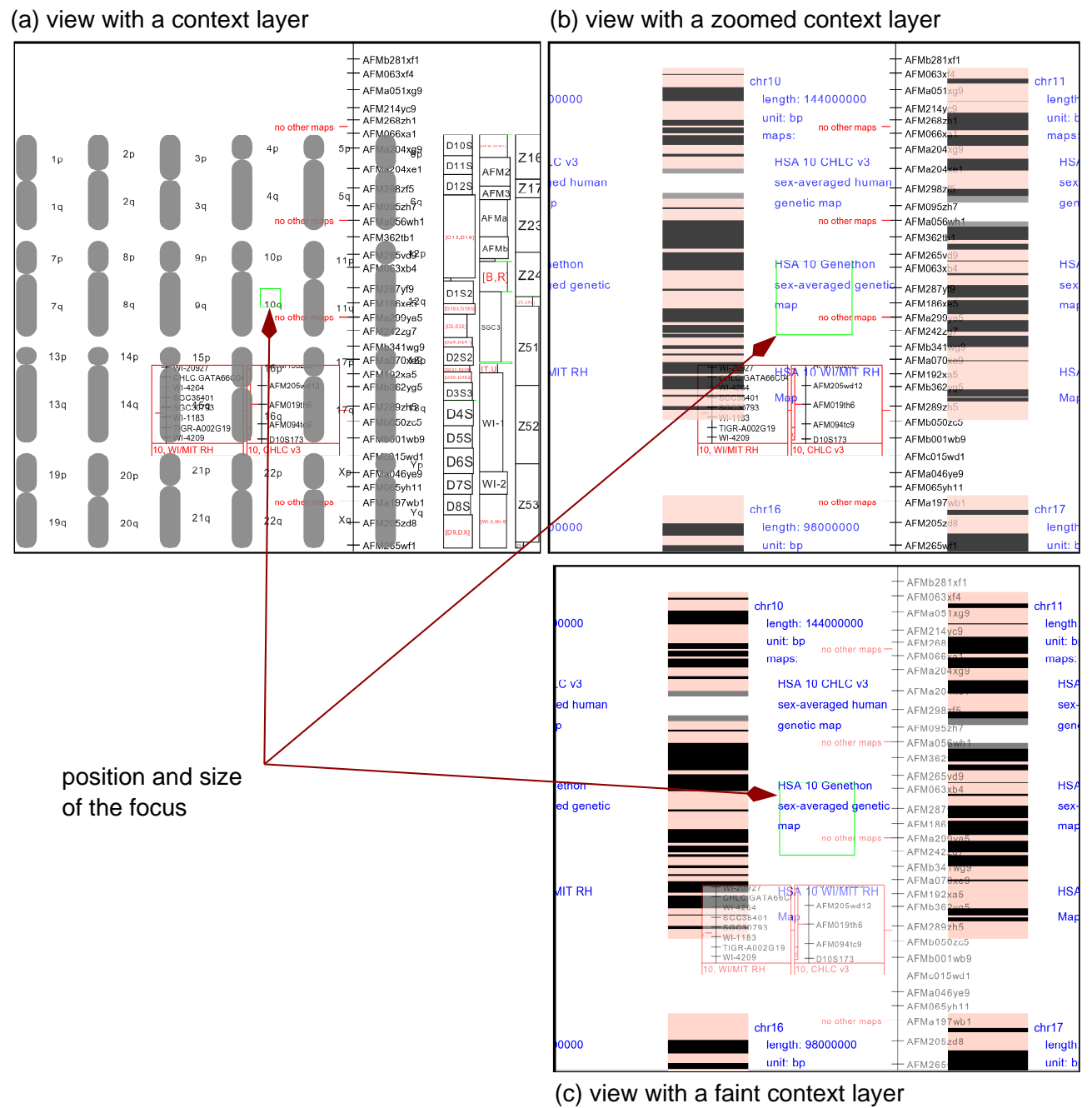


Figure 5.5: Interacting with a context layer

the focus is now bigger relative to the context. Its position, over the name of a map, shows that the focus is showing part of the “Généthon” map. The context can be further dezoomed so as to indicated where the markers currently visible can be found on this map.

5.2.5 No Optical or Positional Deformation

Context layers combine the focus and the context in a way that avoids the positional or optical deformation which often makes images difficult to recognise and understand. Views such as hyperbolic displays or non-linear distortions allow a large amount of context information to be displayed. Users often find it difficult to use this context information because it is severely deformed, either near the edges of the view or just around the focus. The size, form and relative positions of the information are changed. Even more difficult for users is that these transformations are not constant. As users move through the information space and as information moves towards and away from the focus, the deformation changes. The advantage of deformation, that more information can be represented, is retained by the context layer as it associated with a continuous and very fluid control of the scale of this layer.

5.2.6 Distinguishing the Focus and the Context

When two different views are drawn superimposed it is not always easy to see which object belongs to which view; distinguishing the two views may be difficult. When a context layer is being used, one of the two views is transparent. All transparent objects thus belong to one of the views. The most important way of distinguishing the two views comes from the continuous control that the users have over the scale of the context and over the relative levels of the transparency of the two views. As the users change the scale of the context, those objects in the context move and change while the objects in the focus never move. The users can thus easily identify to which views belong the objects. As the users change the transparency levels of the views, those objects that are transparent become paler or more solid. Once again, as the users control the context layer, they can see to which view each visible object belongs.

5.2.7 Context Layer is Quasimodal

The context layer is modal: during its use the display is modified and all mouse movements control the layer. Modal interfaces are often problematic for users as they need to understand how to enter the mode, how to identify when they are in the mode, and how to leave it. These problems were avoided by having the

context layer, and thus the mode, exist only during a single gesture. Once this gesture is over, the interface leaves the mode. This gesture starts when the mouse button is pressed and ends as soon as the button is released. The context layer is thus quasimodal, a term proposed by Raskin (2000), rather than modal. Nielson (1987) uses the term “‘spring-loaded’ modes” and states that they are thought not to be as bad as modes that are entered and left with special commands. Raskin (2000) reports tests that confirm that holding down a key, pressing a foot pedal, or any other form of physically holding an interface in a certain state does not lead to mode errors.

5.2.8 Synergy Between Interaction and Control

The usability of the context layer depends on the user having a fluid control of the layer. The users must be able to create the context layer with a simple gesture and it should disappear as easily. Users need to be able to easily change the scale of the layer so as to see the position of the focus with respect to the global view at many different scales. They also need to be able to easily change the transparency levels of the focus and the context so to switch their attention from one to the other.

People can more easily identify objects when they move, especially when these objects move against a stationary background. The simple act of controlling the context layer, which moves one view in front of a stationary view, is also very important in reducing the interference between the two layers as it allows the users to identify to which view each visible object belongs.

When using a context layer two orthogonal controls are immediately available: the scale of the view shown in the layer and the relative levels of transparency of the context layer and the focal view. These two controls are shown in Figure 5.6. The interactor, a Control Menu, used to provide these controls is described in Part I. Figure 5.7 shows the Control Menu in Zomit and the gesture that created and controlled the context layer through the steps shown in Figure 5.5. The scale of the context layer can be chosen so that the context layer shows any view between the initial view and the focus.

Users can also control the relative transparencies of the context layer and the focus. This allows users to concentrate on either the context or the focus by making the chosen view be drawn solid and the other as transparent as desired.

Both these controls are available at the same time and in the same gesture. Users can change the scale, then change the transparency, and then change the scale again all in the same movement. It is this fluidity of control that should make the context layer usable.

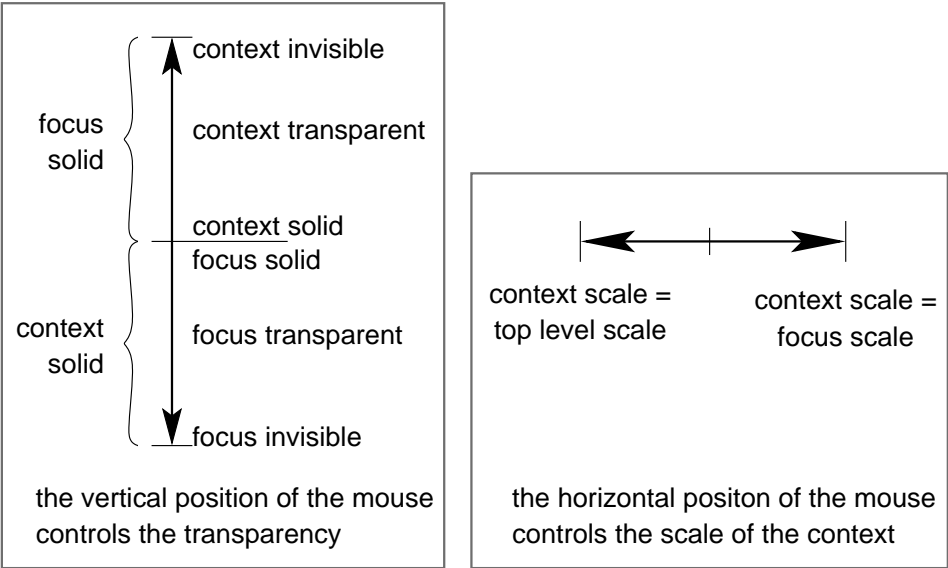


Figure 5.6: Controlling the context layer

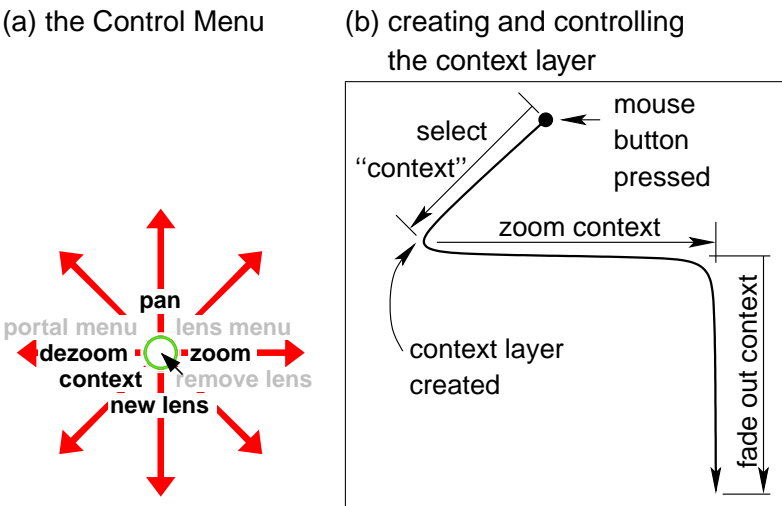


Figure 5.7: Gesture used to create Figure 5.5

5.2.9 Example

When in the situation shown in Figure 5.3, users can ask the ZUI to show the context layer. The context layer contains a contextual view (Figure 5.4c) and is drawn on top of the focus (Figure 5.4a) giving Figure 5.4b.

The position of the focus is indicated on the context layer by a green rectangle. In Figure 5.4a this rectangle covers the text “10q” and tells the user that the focus is showing the chromosome 10. In Figure 5.4b the user has zoomed the context layer so that it shows the names of the genetic maps (the focus never changes during the use of the context layer). The green rectangle showing the position of the focus covers the name of the Généthon genetic map. The focus is thus showing this map. Using the context layer the user has been able to position the focus in two different contexts.

As discussed above, the user can choose to concentrate on either the focus or the context by changing the relative transparency of these two views. Figure 5.4c is similar to Figure 5.4b except that the user is now concentrating on the context and has faded out the focus. The rectangle that shows the position of the focus is always visible and the user can see more clearly that the focus is currently showing the Généthon genetic map.

5.2.10 Similar Techniques

Transparent overview layers (subsection 4.3.7) are a different type of display that differs from ours in that:

- their layer is permanent while ours is a temporary orientation aid;
- the transparency level of their layer cannot be changed by the user;
- their layer always shows the top level view of the information space while ours can be used even if there is no top level view; and,
- their layer can be used to move or modify the objects in the information space while our ZUI does not allow objects to be manipulated in this way.

The context layers’ advantage comes from the coupling or bonding between the interaction (or control) and the visualization.

5.3 History Layer

Context layers allow the user to find the answer to the question “where am I?” Another important question is “how did I get here?” ZUIs need a history so that the

user can return to previously visited regions of the information space and see these regions in relation to the focus and the top level view. We propose a transparent and temporary *history layer* that allows the user to move interactively along the path taken in the ZUI. As with the context layer, the history layer is temporary so as not to overload the screen and it disappears when the user releases the mouse button at the end of the gesture used to create it.

The path taken by the user in the ZUI is a sequence of views of the information space. The first view is the initial (or top level) view (Figure 5.8a) and the view on the screen is the last current view (Figure 5.8e). All the views (called the historical views) seen by users are stored in this sequence. Figure 5.8b and Figure 5.8d are historical views that the user has seen in going from the top level view to the last current view. The history layer is drawn over the top level view (giving Figure 5.8c) and contains a view that can be varied by users from the last current view, via all the historical views in order, to the initial view (and back again). The user can thus interactively “go back in time” and see the evolution of the current view in relation to the top level view. The comparison is done directly because transparent views are used so as to show the top level view and the historical view simultaneously. This comparison is also aided by the rectangles, drawn in two different colours, that show the sizes and positions, relative to the top level view, of the last current view and the historical view.

The history layer is temporary (time multiplexing) and transparent (depth multiplexing).

The interactor used to control the history layer, a Control Menu, is described in Part I. This interactor provides two orthogonal controls, shown in Figure 5.9. The horizontal position of the cursor within the ZUI controls the choice of historical view. When the cursor is towards the right of the interface, the chosen historical view is a view seen recently by the user. As the cursor is moved towards the left, the history view is a view that was seen earlier and earlier in the user’s navigation in the ZUI.

As with the context layer, the relative transparency levels of the history layer and the top level view can be adjusted so users can concentrate on the history layer or on the top level view. As the user moves the cursor towards the top of the ZUI, the top level or context view becomes paler and paler until it vanishes. At the cursor is moved towards the bottom of the interface, the historical view becomes paler until it vanishes. The two rectangles showing the position of the current view and the last current view are always drawn solid and are not affected by the level of transparency.

The current implementation of history overlays requires a top level view. This may not exist in systems where users can dezoom from a view of their own files to a view of, potentially and for example, the whole Internet. The scale of the top level view could also be so different from that where users are currently working

Figure 5.8: Construction of the history layer

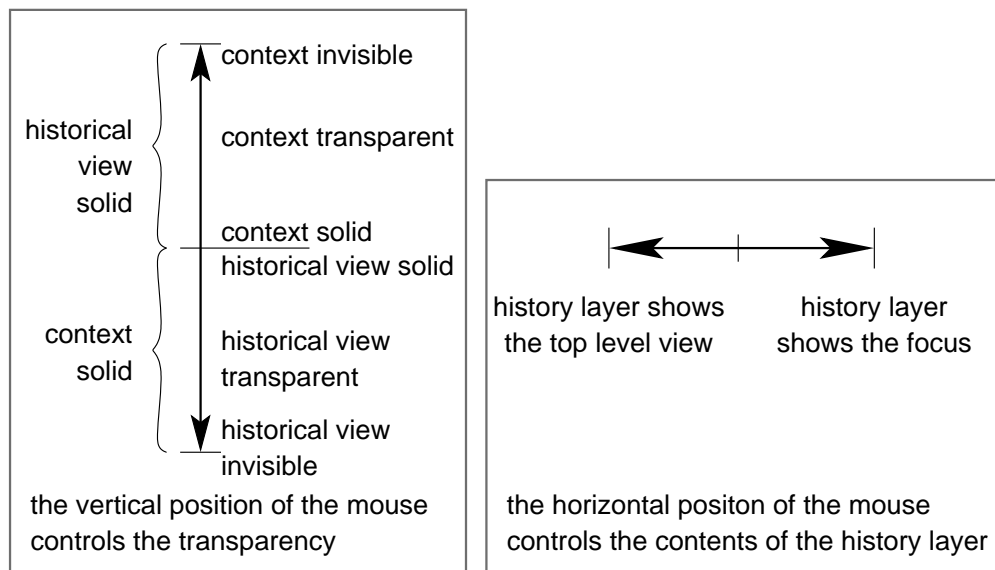


Figure 5.9: Controlling the history layer

that they are unable to see changes to the positions of the current and historical views. Further work should investigate whether the system should choose a different view to replace the top level view and whether (and how) users can control this choice.

5.4 Conclusion

The new context aids presented in this chapter use the space and time multiplexing strategies identified in chapter 4 to facilitate navigation in Zoomable User Interfaces and to reduce user disorientation. These new aids avoid the often difficult to understand positional and optical deformations used by many visualization systems. We have found that the context layer is easier to understand than the history layer. Both these techniques are easier for the user to understand than an onlooker. Observers have a great deal of trouble understanding the relationship between the interaction and the display. An evaluation of the hierarchy tree is presented in the next chapter.

The dynamic nature of these aids is important in two respects. The required contextual information can only be obtained with a continuous interactive use of the aid and this continuous movement is what makes the aid's display easily understandable.

Chapter 6

Zomit: a Development Tool for Zoomable User Interfaces

We developed Zomit,¹ a Zoomable User Interface development tool, in an attempt to solve some of the problems described in subsection 4.4.9. In this chapter we describe the features of Zomit and Zomit's architecture, followed by a comparison with other ZUIs. Zomit implements the new context aids described in the previous chapter and uses the new interaction techniques described in chapter 3. These techniques allow a more fluid and natural control of a ZUI. This improved control of the interface allowed us to develop the new context aids that rely on this new more fluid control for their effectiveness.

6.1 What is Zomit?

Zomit is a tool that allows developers to rapidly develop zoomable user interfaces. It consists of a client and a server library that are used without modification with any virtual world. The communication between the client and the server was optimised so that using the client over the Internet is possible. Zomit provides all the standard functions of a Zoomable User Interface, such as semantic zooming, Magic Lenses and portals, plus our new context aids. The graphical objects available to create the virtual world are simple and limited to lines, rectangles (possibly with rounded corners), text, and JPEG or GIF images.

Zomit was designed to be used with passive databases. The communication protocol and the cache strategy used this design goal to minimise and accelerate

¹The name "Zomit" was chosen because interfaces developed with this tool zoom a lot and because I was wearing my Wallace and Gromit (<http://www.aardman.com/wallaceandgromit/>) T-shirt the day I had to find a name. "Zomit" thus rhymes with "Gromit".

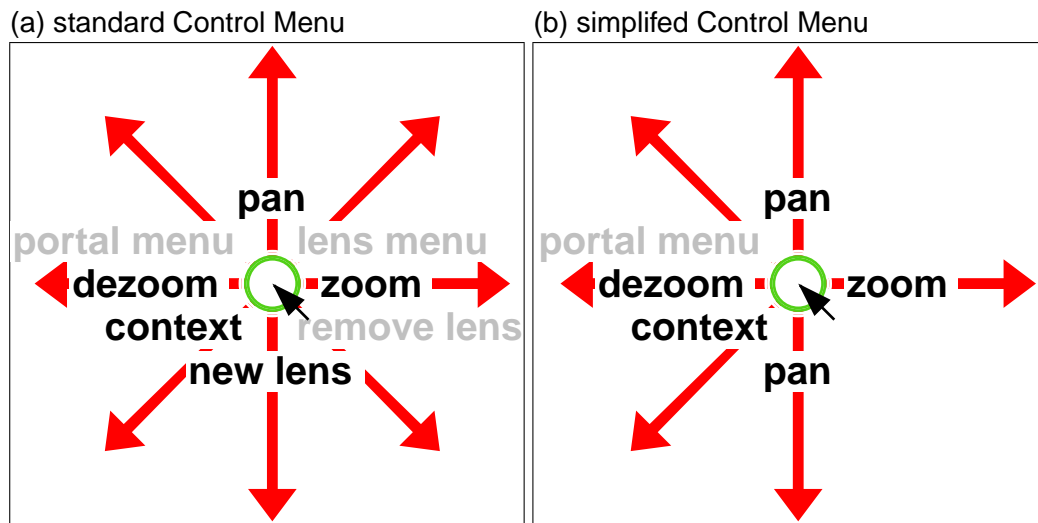


Figure 6.1: Control Menus in Zomit

the exchanges between the server and the client. Zoomable User Interfaces developed with Zomit are used to visualize information spaces but not to modify them interactively.

6.1.1 Navigation and Interaction

The Zomit client (the user interface part of Zomit) uses the new Control Menu described in chapter 3 and incorporates our new navigation aids. Traditional ZUIs rely on the presence of multiple mouse buttons to handle zooming and panning. Other functions, such as creating and moving Magic Lenses and interacting with portals, are implemented by special regions in the interface. For example, the title bar of a Magic Lens is a special region where the gesture that would normally pan the ZUI has a different effect; it moves the lens.

Zomit has a different approach. Only one mouse button is needed (all the buttons on the mouse have the same effect). Multiple functions are accessible from one mouse button via the use of a Control Menu (chapter 3). The top level Control Menus used in Zomit are shown in Figure 6.1. Figure 6.1a is the standard Control Menu. This menu is automatically simplified to that shown in Figure 6.1b if the developer of the virtual world does not declare any lenses.

The same gesture always has the same effect no matter where it occurs in the interface. The gesture that zooms the ZUI will always zoom the ZUI even if it is made over a lens or a portal. This means that users do not have to look at the screen and find a region free of portals and lenses before making the gesture to zoom or pan. This is especially important for portals which arrive over the Internet

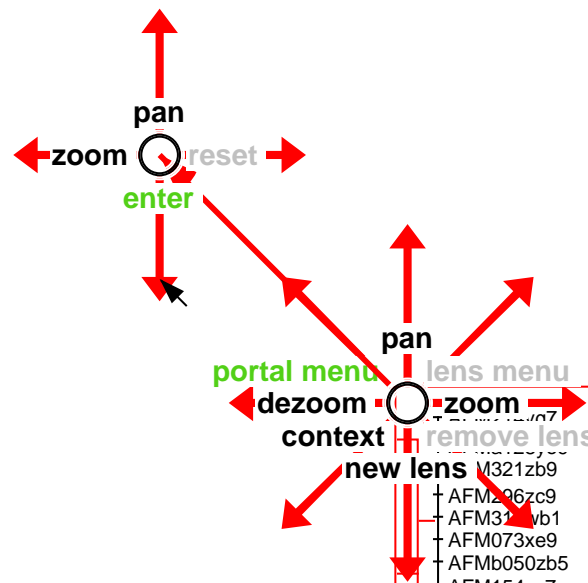


Figure 6.2: Portal sub-menu in Zomit

and can thus be drawn between the moment when users check to see that there are no portals under the cursor and the moment when they start a gesture. Users can operate on lenses and portals by using the sub-menus of the Control Menu. This menu, shown in Figure 6.2, is only active when the cursor is over a portal. The operations are thus separated from normal interaction. Operations on portals are also separate from operations on lenses. It is thus easy to control a portal even when it is covered by one or more lenses.

The pan operation is at the top of the Control Menu. Users must move the cursor up to start the pan operation, and thus move the image up a small amount at the start of a pan. Informal observations indicate that users do not find this difficult or disconcerting.

Earlier menu layouts had the zoom operation only on the right hand side of the Control Menu. This confused many users as they expected to be able to dezoom by moving directly to the left. With the Control Menu layout shown in Figure 6.1, the zoom action is on the left and the right of the Control Menu. Users can thus start the zoom operation with an initial zoom or an initial dezoom. The disadvantage of this solution is that it requires an extra position on the top level Control Menu. Some user studies have reported that users expect the zoom and dezoom operations to be towards the top and bottom of the screen or, equivalently, away from and towards the user. We have not tested this assertion.

Panning and zooming are highly correlated operations. After a zoom, or even during a zoom, users often need to recentre their view of the information space.

An open question is how to combine panning and zooming. Vertical movements of the cursor do not have any meaning during a zoom. It is thus possible to redefine the zoom operation so that a large vertical motion would cause the system to quit the zoom mode and enter pan mode. However it is not possible to then leave the pan mode and return to zooming because during a pan all cursor motions have a sense.

6.1.2 Client/Server Architecture for the Internet

The Zomit system was designed to be usable over the Internet. This means that the user of the system is often a long way from the source of the information used to create the virtual world. The database containing this information is normally much too large to be transferred over the Internet for each user and creating each region of the virtual world may require reading a significant amount of information from the database. The program that reads the database to create the virtual world must thus be able to rapidly interrogate the database. This means that the program must be on the same local network as the database server.

In addition, Zoomable User Interfaces are very interactive programs. The user continuously manipulates the view of virtual world that represents the information space. These interactions must be executed as rapidly as possible and often require the view of the information space to be completely modified. Keeping the response times at acceptable levels requires that the program execute on the user's machine.

These two requirements meant that Zomit program had to be split into two parts, a client and server. The client runs on the user's machine and communicates with the server which executes close to the database server.

6.1.3 Generic Tool

Zomit is a generic tool. The structure of a ZUI developed with Zomit and as used over the World Wide Web as a Java applet is shown in Figure 6.3 (see subsection 6.3.2 for more details on the Zomit client). The same client can be used to visualize any virtual world created with Zomit. The server library is also completely generic. The developer of a Zomit virtual world just has to replace the the HTML page read by the user's browser and the application specific code that connects the Zomit server library to the data source (see subsection 6.3.1 for more details on creating a Zomit server). These components are shown in bold text in Figure 6.3.

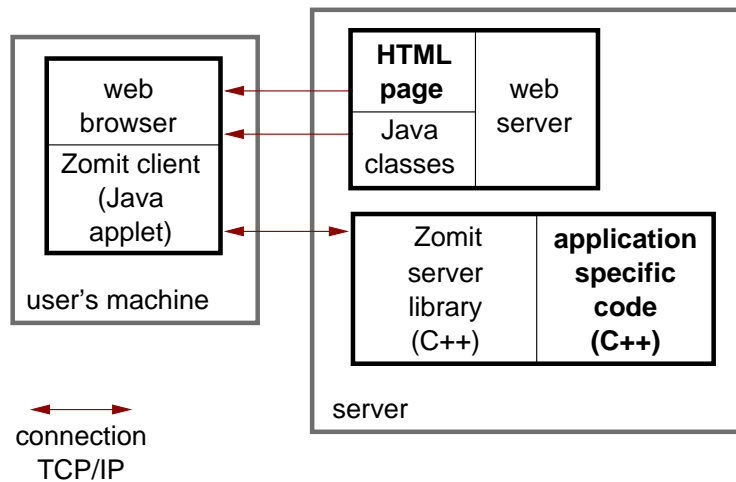


Figure 6.3: Components of Zomit

6.1.4 Creating the Virtual World

A Zomit server consists of a program, written by the developer of the virtual world, linked with the Zomit server library. This program creates instances of classes derived from a base class, *triglyph*,² provided by the library. These instances are registered with the library as covering a region of the virtual world. This base class consists of two methods. The first returns the region associated with this instance when it was registered. The second method, *run*, is a virtual function that the developer must override. When called, the *run* method of these instances generates graphical objects and new instances of *triglyph*. These graphical objects and the new instances must lie wholly within the region associated with the instance.

Figure 6.4 shows a view of a two dimensional virtual world (the *y* dimension has been removed, only the *x* dimension and the scale remain). As users zoom, they move up in the virtual world shown in the figure. Five different viewports into the virtual world are shown as hollow rectangles in the figure and labelled with letters. The viewport always remains the same size so as they zoom to see more details, users see an ever smaller part of the virtual world. Users start at the position *i*. This position is calculated by the client such that the entire virtual world fits into the user's view. Users can then zoom to positions *a* and *b*, and then zoom and pan to position *c*. The object *A* is visible at positions *a* and *b* but has vanished, replaced by objects *B* and *C*, by the time the users have zoomed to position *d*.

Some *triglyphs* are shown in Figure 6.4 as solid rectangles and labelled with

²The reason for the choice of this name is lost in history.

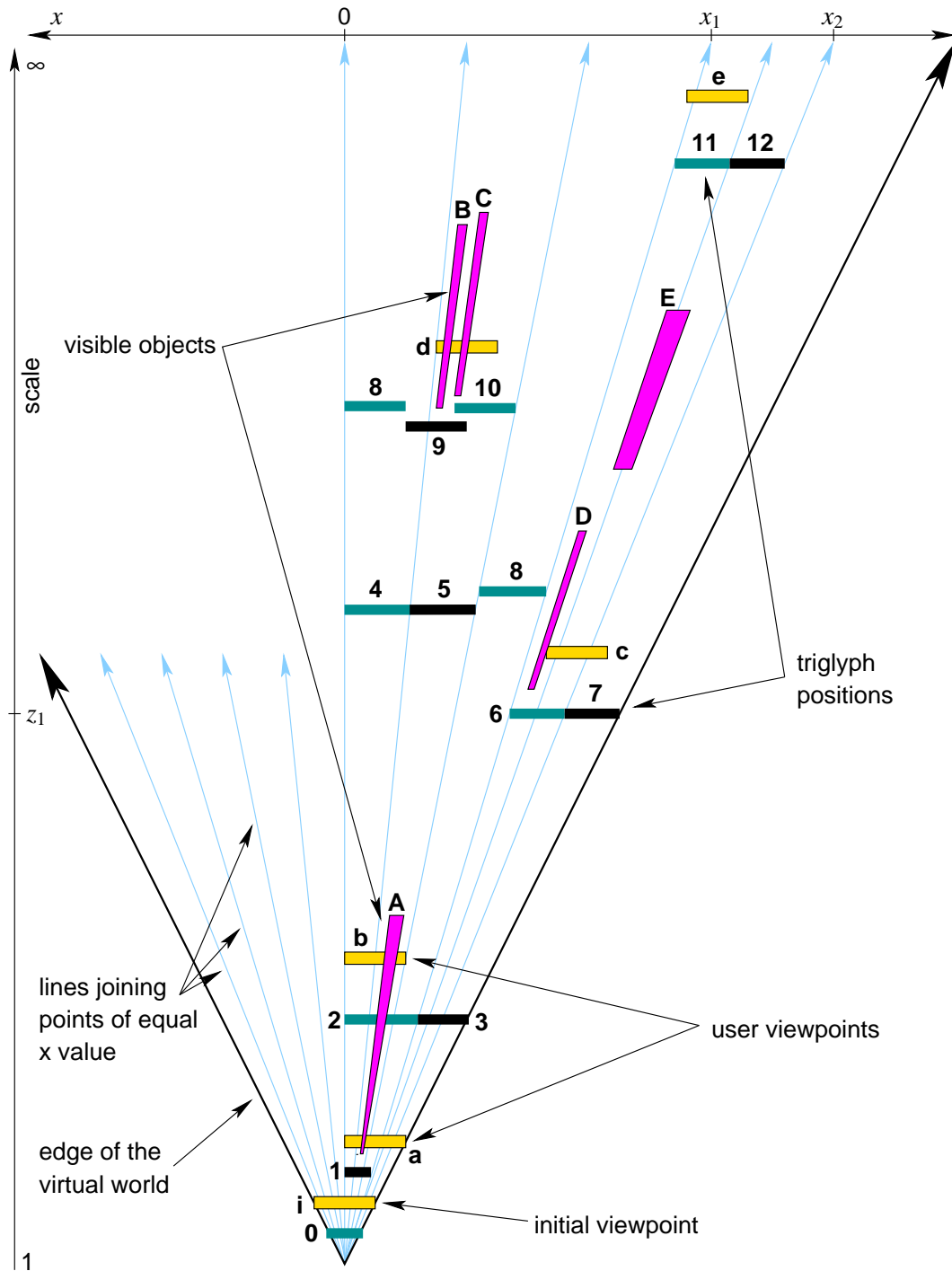


Figure 6.4: Zomit virtual world

numbers. As explained above, triglyphs are defined as covering a region of the virtual world. They cover a rectangular region from a given scale until infinity. As the virtual world in Figure 6.4 has only two dimensions, the rectangle becomes an interval. The triglyph number 6 covers the part of the virtual world between $x = x_1$ and $x = x_2$ and from scale z_1 to ∞ . When called, the *run* method of this triglyph creates graphical objects and other triglyphs in this part of the virtual world. This triglyph could create the objects D and E and the triglyphs 11 and 12. (These objects and triglyphs could also have been created by the triglyphs 3 or 0.)

The *run* method is called when the user enters into the region associated with the triglyph. The triglyph 0 (from Figure 6.4) is called as soon as the ZUI starts. The triglyph 2 is called when the user moves to position b or d. It would not be called if the user moves to position c or e.

6.1.5 Lazy Evaluation

ZUIs are typically used to visualize very large databases. An extremely large number of graphical objects is required to create the virtual world for such a database. Generating all these objects in advance would create another large database and generating them each time a user wanted to visualize the virtual world would take too long. It is thus important to generate the objects in the virtual world only when they are needed.

This mechanism described in the previous section allows the objects in the virtual to be generated only when required, i.e. by lazy evaluation. The developer creates an instance of triglyph (number 0 in Figure 6.4) that covers the entire virtual world. When this instance of triglyph is executed, it will create the objects visible in the top level view, and, new instances (numbers 1 and 3 in Figure 6.4) that will be called if and when the user arrives in the regions of the virtual world associated with the new instances. Objects will thus only be created when required. This allows Zomit to be used to visualize very large databases where it is not possible to create in advance, and store, all the possible objects in the virtual world.

6.1.6 Positioning Regions and Objects

Each region in the virtual world is described by six coordinates: the range covered in the x dimension, the range covered in the y dimension, and the range covered in the scale (or z) dimension. Objects in the virtual world are described using the same coordinates (in this case the scale dimensions indicate between which scale values the object is visible).

Coordinates are Absolute

Objects and regions are always described in absolute coordinates in the server and transmitted as such to the client. The client scales the currently visible objects according to the user's current zoom factor (scale) before drawing them. Objects are visible when the user zooms to the scale from which they are visible. They then grow as the user zooms and vanish when the user's scale is no longer within the range associated with the object. Objects grow as users zoom because the scale changes not because the positions (coordinates) of the objects change.

The developer of the virtual world can freely position the objects in the world. A typical virtual world will however be constructed in such a way that when users zoom on objects, and they vanish, they will be replaced with other objects that describe the underlying information in more detail. The process was graphically illustrated in the section on space-scale diagrams (subsection 4.4.1) and in Figure 6.4. An object could thus be placed at coordinates $(x_1, x_2, y_1, y_2, z_1, z_2)$ to be replaced by, for example, two objects at $(x_1, x_3, y_1, y_3, z_2 + 1, z_3)$ and $(x_3 + 1, x_2, y_3 + 1, y_2, z_2 + 1, z_3)$ (where $x_1 < x_3 < x_2$ and $y_1 < y_3 < y_2$). The different representations of a data item thus remain at the same position in the information space but are represented in more detail as users zoom.

Coordinates are Large Integers

The disadvantage of using absolute coordinates is that the objects that show the virtual world in the least detail have to be large enough to be subdivided into the very large number of objects that describe the virtual world in detail. In Zomit all positions in the virtual world are described using 128 bit integers. This provides sufficient (and uniform) precision for even very large virtual worlds but some care has to be taken to avoid underflow and overflow in intermediate results when performing calculations. Another possibility would have been to use high precision floating point numbers but calculations with floating point numbers is often slower than with integers. The second problem with floating point arithmetic is that floating point numbers do not have a uniform precision. They have a very high precision near zero but the local resolution declines quickly as the value moves away from zero. This resolution granularity could cause problems in accurately positioning objects distant from the centre of the virtual world.

6.1.7 Portals and Lenses

A portal is an object in the form of a rectangle and that includes the coordinates, x , y , and scale, of the region of the virtual world that is to be shown in the portal when it is first displayed. The display of the contents of the portal, and panning

and zooming by the user in the portal, are handled by the client. When necessary the client requests the region to be displayed in the portal from the server. The handling of these requests does not require any code to be written by the developer of the virtual world.

Each Magic Lens and every possible combination of Magic Lenses is a view into a virtual world parallel to the main virtual world. When the developer of the virtual world creates an object or a triglyph, the object or triglyph is placed either in the main world or in the parallel world associated with a Magic Lens. Using this technique, the developer can create objects that are, for example, black in the main world, but have a different colour in a lenses. Another possibility is to change the form of objects when they are viewed through a lens. The effects can be combined when objects are viewed through two lenses. This is discussed in more detail in subsection 4.3.6.

Objects visible in lenses are standard graphical objects with just an extra field which indicates in which lenses they are visible. This allows the client to treat them like any other object and, most importantly, to cache them.

With this type of lens, the developer can create a constant transformation of the objects covered by the lens but cannot create a lens that shows a calculation that depends only on these objects.

6.1.8 Communication Server Side

The server receives requests from the client asking for all the objects that intersect a plane in the virtual world (the viewports shown in Figure 6.4). This plane is described by five coordinates (x_1, x_2, y_1, y_2, z) . When a request arrives the list of those objects already created but not yet sent to the client is examined. Those objects that intersect the plane are sent to the client and deleted from the server. The list of instances of triglyph that have not yet been called are then examined. Those whose regions intersect the plane are called. The resulting objects are either sent directly to the client if they intersect the plane, or stored. The procedure is then repeated with those instances of triglyph generated by the instance of triglyph that was called.

The client sends two types of request to the server. The first asks for the objects in a (small) region of the virtual world that corresponds to the contents of a portal or a lens. These requests are processed in the order in which they arrive. The second type of request indicates that the user has moved (by zooming or panning) and that all previously received requests (including those of the first type) should be abandoned. The queue of incoming requests is thus examined, each time an object is sent to the client and before calling a triglyph, to see if there is a request that should cause the current request to be abandoned. This avoids sending graphical objects that are no longer required or unnecessarily executing

triglyphs.

6.1.9 Communication Client Side

Each time the user changes position in the virtual world (either by panning or zooming), the client sends the user's new position in the virtual world to the server. The client also sends a request for the objects in a region when it needs to draw the contents of a portal or a lens. These requests for small areas of the virtual world are repeated when the user zooms or pans in a portal or moves a lens. As described above, the server responds with the objects (that have not already been sent to the client) that are visible at the requested positions. The client does not wait for the requested objects to arrive from the server (in fact it does not even know if any objects will arrive). It immediately draws the objects that it already has and draws any new objects when they arrive (if the user has not moved to a position where they are no longer visible). All the objects received from the server are stocked for reuse when the user returns to a position where they are visible. This method ensures that only the display of as yet unseen objects is delayed by the latency of the network connection with the server. All other interaction is carried out locally in the client. As objects are visible through a range of scales, when the user zooms it is probable that at least some objects already received from the server will still be visible at the new scale. These objects are thus drawn immediately, thus improving the perceived responsiveness of the system, and as the new objects arrive they fill in the empty spaces.

There is currently no indication transmitted from the server to the client saying that a request has been satisfied. This should be added to the protocol so that the client can tell the user when all the required objects have been received from the server and thus the current view of the virtual world is complete. This lack has not been a problem for our tests because over local Internet connections the objects arrive quickly and, more importantly, in a continuous stream. Once the screen stops changing, the user can tell that all the objects have arrived. This indication is required when the connection has long and random delays.

6.1.10 Exchanges Between the Server and Client

Figure 6.5 shows the exchanges that occur between the Zomit server and the Zomit client.

After Initialisation the Exchanges are Asynchronous

After handshaking with the client and testing that ensure that the client and the server can communicate binary data correctly, the server tells the client the size

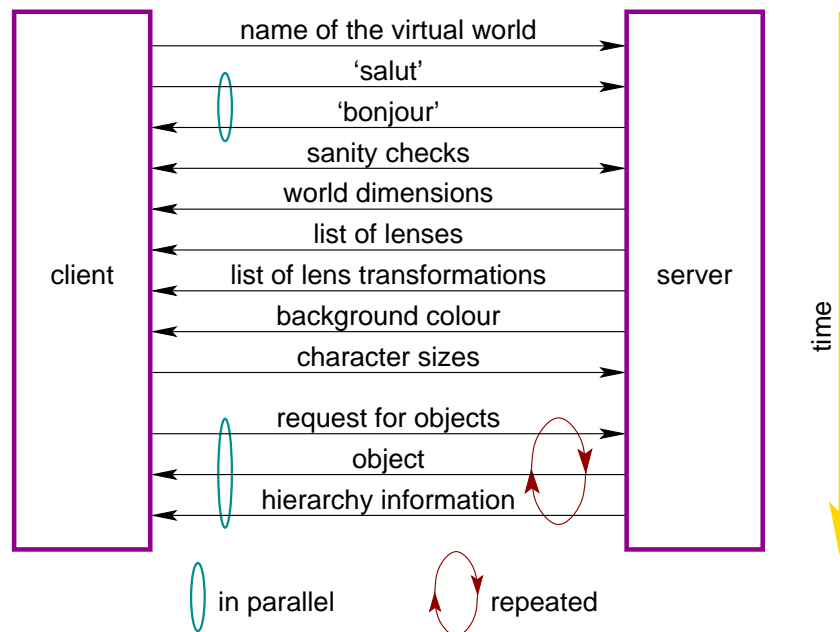


Figure 6.5: Exchanges between the server and client

of the virtual world. The client uses this size to calculate the user's initial scale (or level of zoom) that causes the entire virtual world to be visible. The names of the available lenses are then sent, followed by the colour of the background of the virtual world. The client then sends information on the sizes of the characters that can be drawn by the client. This information is used by the server to calculate the bounding boxes of strings of text and is also available to the developer of the virtual world. Once this information has been sent, the exchanges become asynchronous, the client asks for objects without waiting for the reply, and the server replies as soon as it can.

Exchanges Optimised for Slow Links

Other than the transfer of the client itself, the sending of objects from the server to the client accounts for most the bytes exchanged. The objects are sent in a binary format that corresponds to the format of integers (and strings) in the Java virtual machine. (The Java virtual machine is the program that executes the Java code. It is part of the Web browser used to run the Zomit client.) These objects are thus small and easily unpacked by the client. These objects are sufficiently small that Zomit can be used (but not downloaded) over a mobile telephone connection at 9600 bits/second. Zomit has also been successfully used over long distance Internet links such as that between Paris in France and Palermo in Italy. The capacity

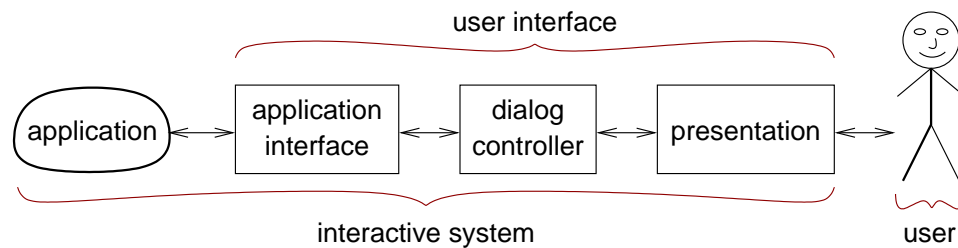


Figure 6.6: Seeheim model

(the number of bytes per second) of the link is thus relatively unimportant. Any delays that the user might see in the drawing of as yet unseen objects is caused by the calculation time on the server or, more often, the latency (the time take for a byte to travel from the server to the client) of the Internet link.

As previously explained, a further optimisation is that objects are send to the client only when they are required.

6.2 Architecture Analysis

This section describes various architecture modelling techniques and then uses one of them to analyse Zomit and a number of other ZUIs.

6.2.1 Architecture Modelling Techniques

There are essentially two different methods of modelling interactive systems (Baudel and Beaudouin-Lafon, 1998).

Seeheim

The first method of modelling interactive systems can be called “vertical”. The modelling techniques that are descended from the Seeheim model (Pfaff, 1985), named after the town where the workshop that invented this model was held. The Seeheim model decomposes the entire user interface part of an interactive system into three components: the interface with the application, the dialog controller and the presentation component. Figure 6.6 shows these three components (Coutaz, 1993). This model emphasises the form of an interactive system and ignores the system’s dynamics. This is unsatisfactory in computer-human interaction where the input and the output evolve together and where multiple input channels may be used in parallel.

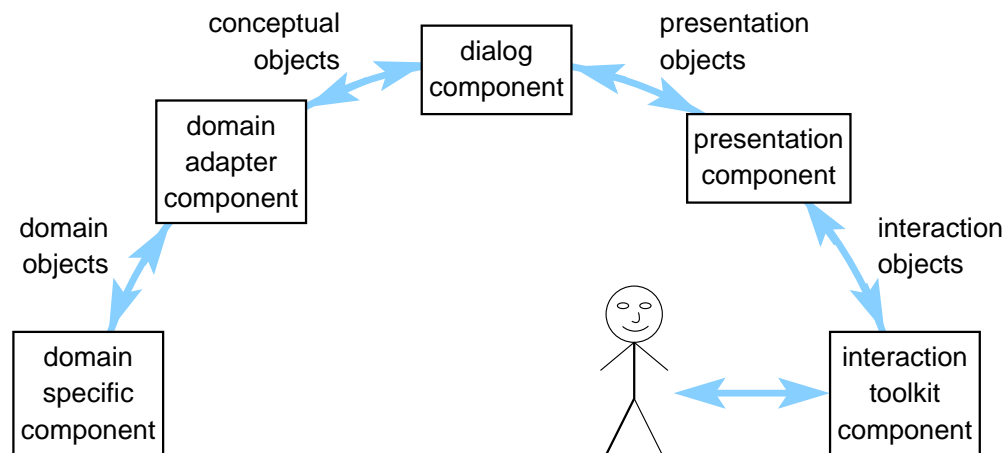


Figure 6.7: Arch model

Arch

The Arch model (Arch, 1992) was created as an improvement on the Seeheim model and with the aim of helping developers understand the advantages and disadvantages of different run-time architectures. The types of information that cross the component boundaries have been made explicit in the model. There are five layers in the Arch model (Figure 6.7). The domain specific component corresponds to the Application in the Seeheim model, the domain adapter component to application interface, and the dialog component to the dialog controller, while the Seeheim presentation component has been split into two parts (Coutaz, 1993). The interaction toolkit implements the interaction with the user and is often provided by an external toolkit such as Motif (The Open Group, 1997). The presentation component is a mediator between the dialog component and the toolkit, and insulates the dialog component from changes in the toolkit. The Slinky meta-model, a generalisation of the Arch model, is a modelling framework from which Arch models, specialised for a particular case, can be derived. A specialised Arch model might remove part of the presentation component if the efficiency is more important than the ability to changing toolkits.

MVC

The second method of modelling interactive systems can be called “transversal” and is exemplified by the MVC (model-view-controller) architecture from Smalltalk (Olsen, 1998). The components of this model are shown in Figure 6.8. MVC models a system by saying that each object in the system has a view, a controller and a model. The view must provide the graphical representation of the

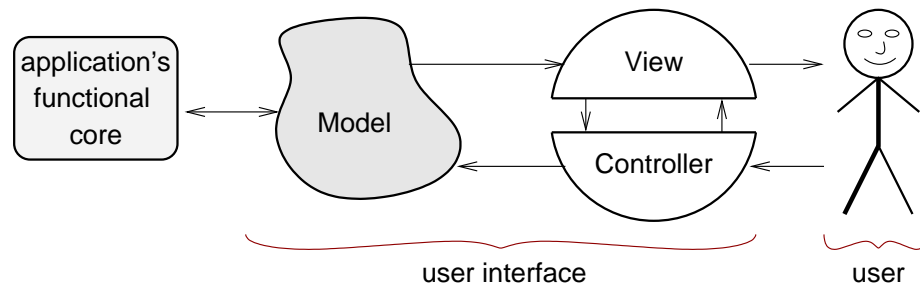


Figure 6.8: MVC model

object that the user can manipulate. The controller receives all of the user's inputs and is responsible for updating the view and calling the objects in the model to make the required changes. The controller may need to interrogate the view to retrieve information required by the objects' callbacks. The MVC model was successfully used in the Amulet project (Myers et al., 2000). This model is called "transversal" because each MVC module is responsible for the presentation, user dialog, and semantics of part of the overall application. A typical user interface system consists of a large number MVC modules. This can make implementing the constraints between the modules and creating a global interaction structure difficult. The use of this model can thus become unwieldy when the dialog model is complex. In addition the model cannot represent the application as a whole—each part of the application is modelled independently.

PAC

As with the MVC model, the PAC (Presentation, Control, and Abstraction) model (Coutaz, 1990) decomposes each class in the application into three components: Presentation, Control, and Abstraction (Figure 6.9). This model differs from the MVC model in that all the interaction with the user (that is the view and the controller in the MVC model) is combined into a single component, called the presentation. The control of the module is now explicit: the Control component is responsible for linking the presentation and, via the abstraction, the application. The PAC model is recursive and the Control component is the liaison with the module above this one in the module hierarchy, and with those modules below it.

PAC-Amodeus

The PAC-Amodeus (Assimilating Models of Designers, Users and Systems) model is a refinement of the Arch/Slinky model. The Arch/Slinky model does not formalise the organisation of the dialog controller. In PAC-Amodeus, this component

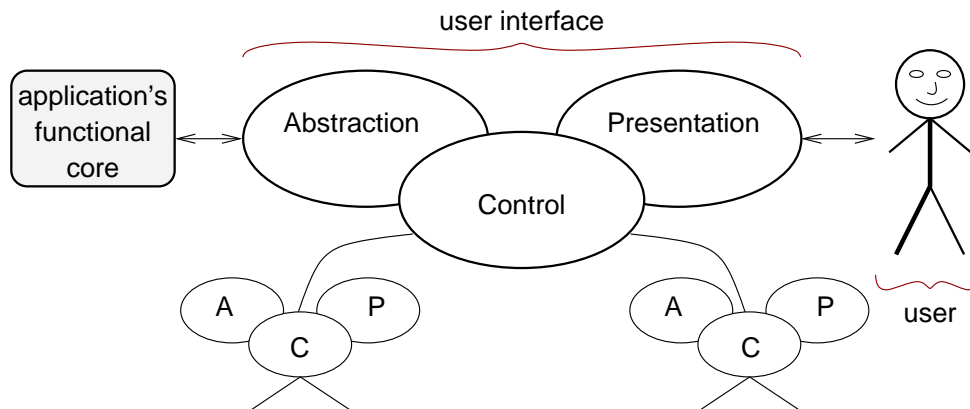


Figure 6.9: PAC model

is defined as a hierarchy of PAC agents (Nigay and Coutaz, 1992). This hierarchy agents must sequence the tasks to be performed in the application as well as translating the data in the system between the different formalisms of the domain objects and the presentation objects. The PAC-Amodeus model was used to model a multimodal airline travel information system (Nigay and Coutaz, 1995). This model is rich enough to be used to describe a complex system that can merge asynchronous spoken commands, typed natural language, and keyboard input into a single user request.

6.2.2 Zomit Modelled by PAC-Amodeus

Figure 6.10 shows how the Zomit client/server architecture can be modelled using the Arch model.

Domain Specific Component

The domain specific component is part of the server that runs on the server machine. This machine stores the data used to generate the virtual world. The domain specific component reads the data used to create the virtual world. In the case of ZoomMap, this component reads the HuGeMap database and creates the domain objects that are passed to the domain adapter. In the CDI application (section 7.2), the domain specific component reads the XML data files to create the domain objects. The domain objects are the class instances that are called by the domain adapter component.

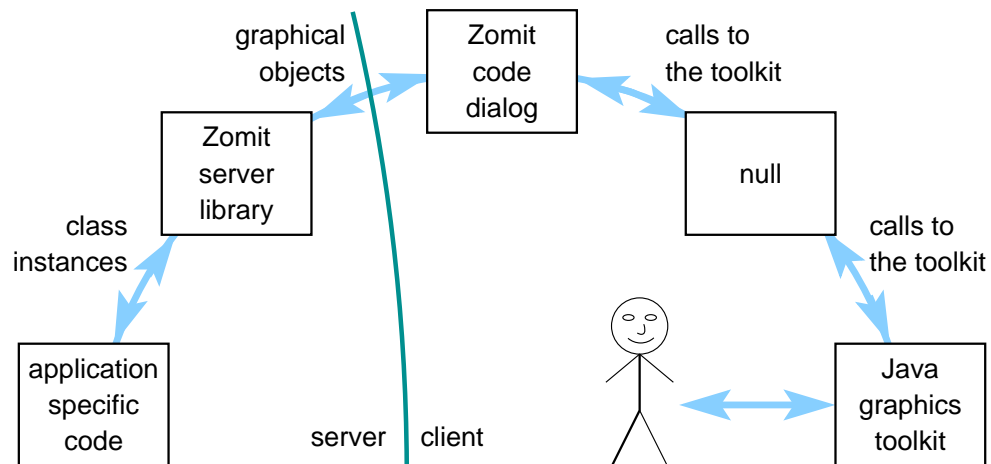


Figure 6.10: Zomit modelled by Arch

Domain Adapter Component

The domain adapter component is part of the server that runs on the server machine and consists of the Zomit server library that translates the classes containing executable C++ code into the simple graphical objects that are transferred over a Internet connection to the dialog component. These simple graphical objects are the conceptual objects that the dialog component stores and uses to control the interaction.

Presentation Component

As indicated in Figure 6.10 there is no presentation component in Zomit. The Zomit client executes on the user's machine which is of unknown computing power. The client must remain rapid even on machines with limited resources. We thus decided for reasons of efficiency to avoid the performance loss that would be incurred by the addition of an extra layer, the presentation component, into the Zomit client.

Interaction Toolkit Component

In addition, the presentation component exists to allow the interaction toolkit, and thus the environment (the X Window System versus Windows, for example) to be changed without requiring the dialog component to be rewritten. This possibility of changing toolkits was not important in this project and Java already provides environment independence. What would have been interesting would have been to replace Java by an another language such as C++. As the dialog com-

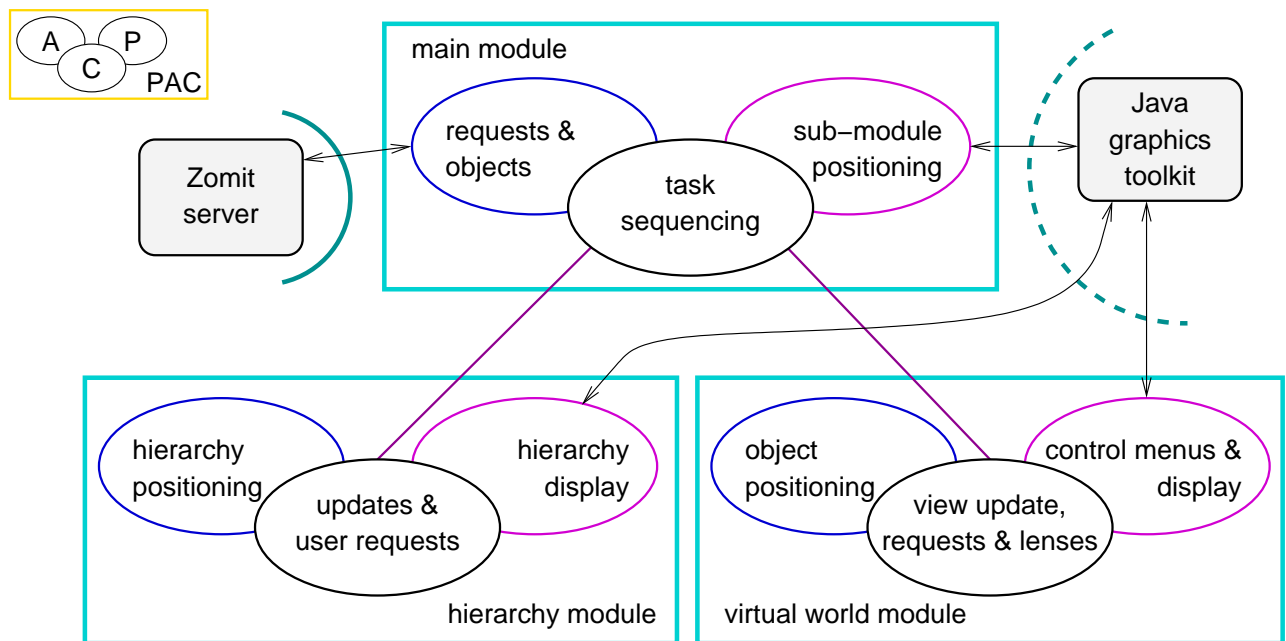


Figure 6.11: Zomit client modelled by PAC

ponent is written in Java it would have had to have been rewritten even in the presence of a presentation component.

Dialog Component

The dialog component can be modelled by PAC as shown in Figure 6.11. This figure uses the same positions for the three components of the PAC module (from left to right, abstraction, control and presentation) as in the Figure 6.9.

Main Module The presentation part, or “axis” (Coutaz, 1993), (labelled “sub-module positioning”) of the main module positions the two subordinate modules (the hierarchy and virtual world modules) and handles resize requests from the user. The abstraction axis (labelled “requests & objects”) handles all the communication with the Zomit server and contains the reader and writer threads described in Figure 6.13. The control axis (“task sequencing”) receives requests from the abstraction axis and passes them to the two subordinate modules so that they can update their views of the virtual world. It also receives requests from the subordinate modules indicating that the user has moved in the virtual world. These requests are passed to the other subordinate module so that the two views of the virtual world stay synchronised and to the server, via the abstraction axis, so as to receive any as yet unseen objects visible at the user’s new position.

Hierarchy Module The control axis of the hierarchy module (“update & user requests”) receives hierarchy information from the main module and uses its abstraction axis (“hierarchy positioning”) to convert this information to screen coordinates. It then asks its presentation axis (“hierarchy display”) to update the user’s view. User interaction with the view of the hierarchy is treated similarly but in the other direction. The control axis receives notification of the user’s demand from the presentation axis and uses the abstraction module to translate the demand to object coordinates before sending it to the main module.

Virtual World Module The function of the control axis in the virtual world module is similar to that of the control axis in the hierarchy module. It asks for and receives objects from the main module, uses its abstraction module (labelled “object positioning”) to translate these objects to screen coordinates (which are different from those in the hierarchy module) and then calls its presentation module (“virtual world display”) to update the user’s view of the virtual world. In response to mouse movements and clicks, the presentation axis displays a Control Menu and interprets the user’s use of the menu. The presentation axis then passes the user’s command to the control axis, which once again uses the abstraction for the conversion to the object’s coordinates, before passing the user’s new position to the main module. If the user asks for the creation of a lens, the control axis asks the presentation module to draw it, and, after translation by the abstraction, asks the main module for lens’ contents.

6.3 Implementation

Zomit was implemented using the models described in the previous section and taking into account the limitations imposed by its execution environment: large databases, long distance access and a slow client.

6.3.1 Server

As the server executes in a known environment, the server machine, it is possible to use a compiled programming language. This also results in a faster server. The Zomit server library is thus written in the C++ Programming Language (Stroustrup, 1997) and consists of almost 5000 lines of C++ code.

The main thread in the server executes the triglyphs, the code specified by the developer of the virtual world. There are two other threads that handle the communication with the client in order to avoid the server or the client blocking when reading or writing the network connection. These three threads are shown in Figure 6.12. Sending objects to the client is handled in a separate thread so that

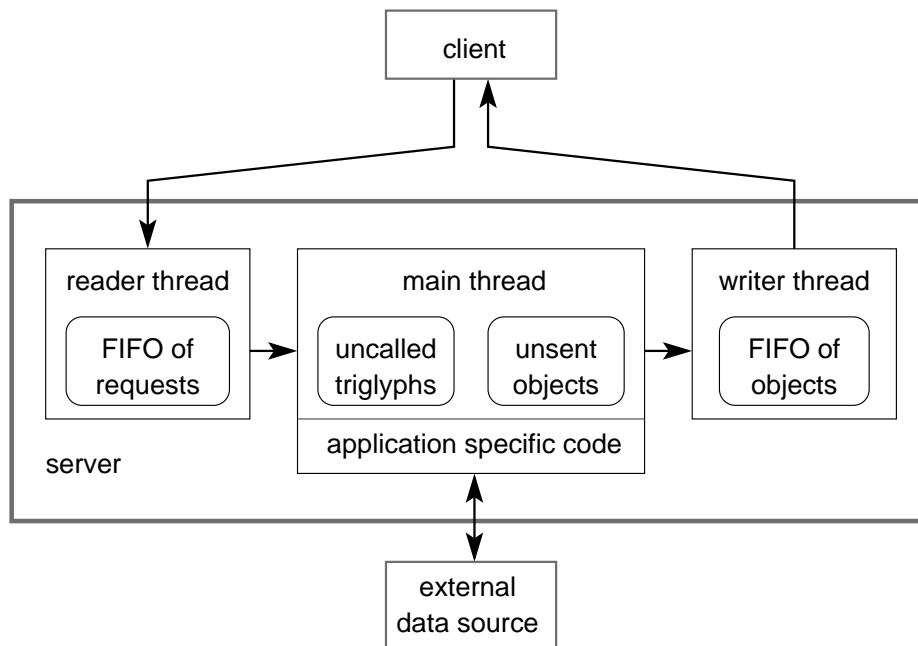


Figure 6.12: Threads in the Zomit server

the main thread is not blocked if the network connection is slow and thus unable to accept the objects as fast as they are generated. Reading the connection from the client is also handled by a thread so that the connection from the client is emptied as soon as possible (even when the main thread is busy). Emptying the connection from the client is important to avoid the client becoming blocked when writing to the server. The coordination and communications between the three threads is handled via mutex locks and shared memory.

6.3.2 Generic Client in Java

The client was written in Java (Arnold et al., 2000) so that it can be downloaded and executed in a standard World Wide Web browser. It has been successfully used with Netscape and Microsoft Internet Explorer running on the Sun Solaris, Linux, MacOS, and Windows platforms. The use of Zomit does not require any installation by the user other than the installation of a standard browser. The Java program, almost 6000 lines of source which compiles to 106 000 bytes of Java code, is stored in a Java archive and is thus downloaded by the Web browser in a single Web connection. As it is completely generic it can be cached by the user's browser or by standard site-wide Web page caches; subsequent downloads are thus accelerated. When downloaded by a browser a Java program is an applet and

can only contact servers on the same machine from which it was downloaded. The Zomit client can also be installed and executed as an application using Sun's Java Development Kit (JDK). In this case the Java archive only has to be downloaded once and can be used to communicate with any Zomit server on any machine. The Java client is a "thin" client: everything that can be done by the server is done by the server so that the client is as small and fast as possible.

When used as an applet, the number of the TCP/IP port of the Zomit server is specified as parameters to the applet in the HTML code that calls the applet. (The machine is always that from which the HTML code was downloaded.) It is thus possible to have a number of independent Zomit servers on the same machine. A number of different Zomit applications can also share the same port on the same machine as the handshaking of the client/server protocol includes the name of the Zomit server to be executed. This name is also specified as a parameter in the HTML code. When used as an application, the user specifies the name of the server machine, the name of the server, and the port as arguments to the application.

One of the aims of Java is to be "Write Once, Run Anywhere" (a Sun Microsystems trademark). This however turned out to be far from the case. Different versions of Java provide different functionalities. The printing functions and some graphics capabilities are not available in the earlier versions of Java (known as Java 1) but only in Java 2. The standard Web browsers only provide Java 1 functions and this has to be detected (without provoking an error) and compensated for at run time. However the biggest problem came from the different implementations of the same version of the graphics libraries. The differences between the Java 1 graphics libraries were such that continuous testing was required to ensure that Java applet produced the same display in the different browsers.

The three threads that execute in the Zomit client are shown in Figure 6.13. The reader and writer threads exist in the client for the same reason that they exist in the server: to avoid the main thread blocking when reading or writing the network connection.

The user's position and scale are restricted so that they remain within the virtual world. If users enter a region where there are no visible objects (or they become lost) they just have to dezoom until they find known objects. In the worst case this will be the initial view of the world. The designers of the virtual world should take care to minimise or eliminate empty regions in the virtual world. This can be done filling these otherwise empty spaces with "sign posts": arrows and text saying "go this way to find interesting objects".

6.3.3 Graphics in Java

The virtual world shown in Zomit is graphically very simple. The user interface consists essentially of two Java Canvas widgets from the standard Java 1 AWT

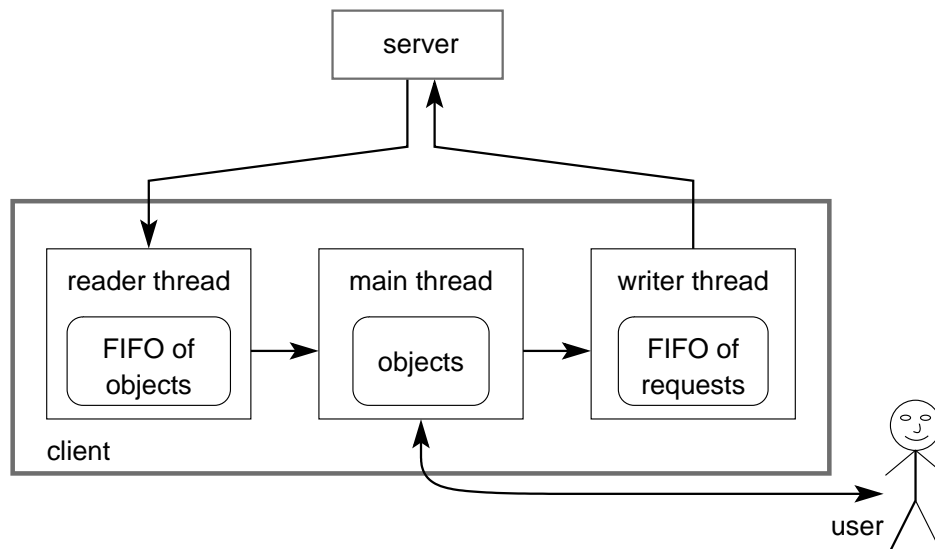


Figure 6.13: Threads in the Zomit client

graphics library. Other widgets are used to position the two main Canvas widgets. These widget are very simple: the programmer is responsible for drawing the contents of a Canvas widget and for redrawing the contents when required (by part of the widget being covered and then reexposed). Only lines, arcs, text, JPEG images and GIF images are drawn. The use of graphical operations was restricted to this set in order to minimise the time required to draw each view of the virtual world. Users move rapidly through the virtual world so it is important that the frame rate, the number of views of the virtual world that Zomit is able to draw every second, be as high as possible. The limited set of operations also reduced the number of compatibility problems between different implementations of the AWT graphics library. JPEG and GIF images are asynchronously scaled to the correct scale by the AWT graphics library. Once scaled they are drawn at the correct position on the screen. The AWT graphics library stores the scaled images so that if they need to be redrawn at the same size, it is not necessary to recalculate the scaled image.

The biggest remaining problem was in the handling of the redrawing of the virtual world during panning and zooming. Zooming leads to the entire contents of the screen being redrawn with each movement of the mouse. Even with the restricted set of graphics operation described above, redrawing the screen can take a significant part of a second on less powerful machines. This operation thus required the use of double buffering to avoid excessive screen flicker. Double buffering, drawing the new image into an off-screen buffer and then copying the off-screen buffer to the screen, increases the time required to update the screen. It

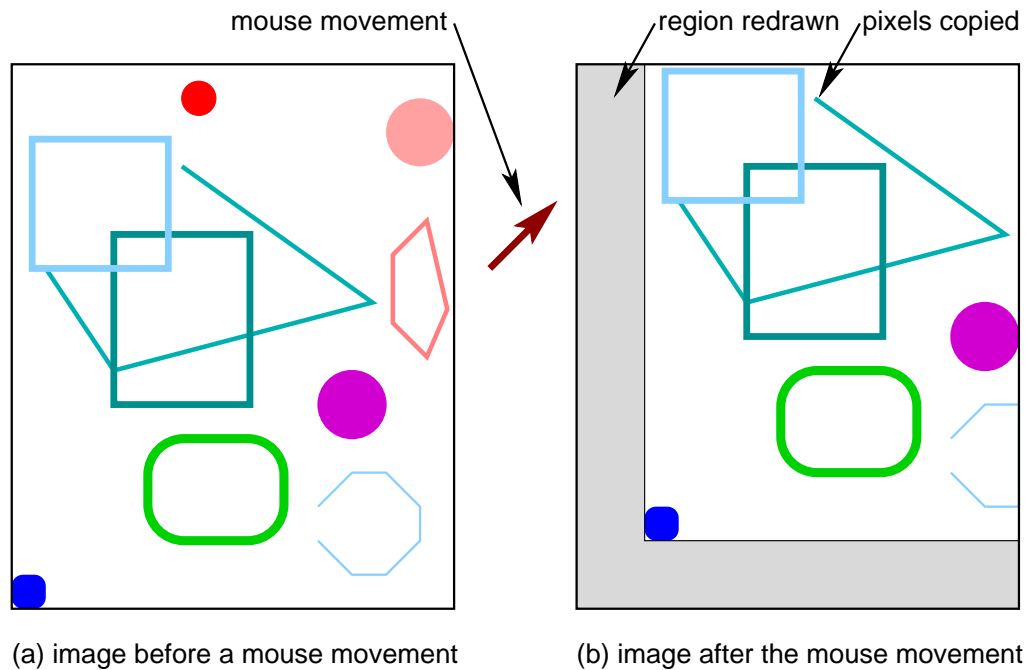


Figure 6.14: Redrawing after a mouse movement (up and right) during a pan

is thus necessary to adapt the frame rate to the time required to redraw the screen. Rather than trying to redraw the screen each time the user zooms a fixed amount, Zomit (during a zoom operation) reads the new mouse position, and thus the new scale, after each redraw, and then redraws the screen with this new scale. This prevents Zomit from “lagging behind” the user.

It was possible to avoid double buffering (and the associated extra graphics operations) during a pan by shifting, via a copy, the on-screen image the required amount and then redrawing the newly empty regions (Figure 6.14). As with the zoom, no attempt was made to redraw the screen each time a mouse movement event is received. After each redraw, all the untreated mouse movements are read, the new position of the mouse calculated and the image “jumped” to the new position. Once again this stops Zomit lagging behind the user. A modern machine is however sufficiently rapid to handle every mouse event as it occurs and to update the screen after each mouse movement.

The rapid redraws described in this section only apply to regions that contain objects already received by the client. When the user visits a new area of the virtual world, the screen will first be redrawn with the currently available objects. The remaining objects will be drawn as they arrive from the server.

Transparent Layers

The development of the new context aids (that use transparent layers) described in section 5.2 and in section 5.3 required the use of Java 2. Java 2 includes new graphics capabilities available in the Java2D extensions to the AWT graphics library. Zomit uses the alpha blending capabilities of Java2D to draw the transparent layers. These extensions are not available in Java 1 which is the version of the Java graphics libraries included in the standard Web browsers. As described above, it was thus necessary to avoid calling the Java2D extensions (or even using class files containing calls to these extensions) when a Web browser was being used. When the Zomit client is being used as an application, the extensions are called and, if available, used. In Web browsers, or if the Java2D extensions are not available, the transparent layers are simulated using the *xor* drawing function of the Java 1 libraries. The *xor* drawing function produces some unexpected colours but it is usually possible to understand the idea behind the new context aids. The *xor* drawing function also has the advantage of being very significantly faster than the alpha blending functions.

The Java2D extensions are also used to create Postscript images of the screens in Zomit. This allows images of Zomit included in reports to be drawn with the full resolution of the printer used. The printing capabilities of Zomit are only available when the client is run as an application in a Java2D capable Java Toolkit.

OpenGL Capable Graphics Cards

Personal computers are now often sold equipped with OpenGL capable graphics cards. These cards include hardware that can draw complex three dimensional scenes very rapidly and without assistance from the computer central processing unit. The Java3D graphics library uses the OpenGL capabilities of graphics cards if available. We considered replacing the AWT graphics library with the Java3D graphics library but decided that the gain in speed was not certain in this application. Zomit does not use the complex three dimensional graphics objects for which OpenGL was designed and modern graphics cards also have hardware support for the simple two dimensional graphics operations that Zomit does use. OpenGL would certainly have been an advantage in the display of our transparent context aids. We decided however not to implement an OpenGL version of Zomit because the implementation difficulties, the subsequent need for users to have an OpenGL card and a Java library that supports OpenGL. The Java libraries in standard web browsers do not support OpenGL.

Further discussion of the advantages and disadvantages of the use of OpenGL can be found in Beaudouin-Lafon and Lassen (2000).

6.4 Other Zoomable User Interfaces

In this chapter we have described the goals of Zomit and the client/server structure that was developed to satisfy these goals. Other ZUIs had different desiderata and thus have different implementations. We discuss some of these other systems.

6.4.1 Pad

Pad (Perlin and Fox, 1993) was the first ZUI. Its goal was broad: to create an infinite two dimensional information space shared among remote users. While the goal of creating a shared information space was not realised, it is the direct ancestor of a number of other ZUIs.

The display of objects in Pad differs from that in Zomit. The Pad Surface contains Pad Objects that are positioned on a fixed region of the surface. These Pad Objects are high level entities with which the user can interact. Examples are editable text files, a clock or a calendar. When the region associated with a Pad Object is visible, the object receives information about what parts of it are to be visible and at what scale. The object then generates the display items (visible graphics) that are to be drawn. Each display item has a range of magnification (scale) outside of which it is invisible. It can also have a transparency range which is a range of magnifications outside which the item is transparent. This allows graphics to fade in and then fade out as users zoom.

The Pad system consists of three layers. The base layer is a real-time display engine written in C++. Pad Objects are written in the Scheme language. This code is interpreted and communicates with the display engine via a C++ interface. Pad ran under the X Window System and MS-DOS.

6.4.2 Tabula Rasa

Tabula Rasa (Fox, 1998) is a continuation of the Pad project written by one of the developers of Pad. The main difference between the two systems is that the interpreted Schema code in Pad has been replaced by compiled Schema code. Tabula Rasa only works under the X Window System but a portage to Windows 95/NT was envisaged.

6.4.3 Pad++'s Architecture

As with Zomit, Pad++ was developed as a tool for developing ZUIs (Bederson et al., 1996). The toolkit is written in C++ and uses the X Window System (Scheifler et al., 1992) on workstations running UNIX (or Linux). (Further information

and the program's source are available for download at the Pad++ home page at the URL <http://www.cs.umd.edu/hcil/pad++/>.)

Monolithic Architecture

Pad++ has a single program with monolithic architecture. This single program reads the data source, generates the code that will be called to draw the virtual world, draws the user's view of the virtual world, and handles all the interaction with the user. The only possibility for separating the execution of Pad++ from its display is via the standard X Window System display protocol. This separation would be at the interaction toolkit component in the Arch model (subsection 6.2.1). This protocol, where every user action and all screen updates pass by the network, was designed for use on a single workstation or over local area networks, not for complex high-speed interactions over long distances. This is in contrast to Zomit which has the separation between the calculation and the display at a different level in its architecture (Figure 6.10).

Virtual World Described Using Tcl/Tk

The C++ "substrate" of Pad++ is linked with the Tcl/Tk scripting language (Ousterhout, 1994). The developer of a virtual world in Pad++ describes the virtual world using Tcl/Tk. Tcl/Tk is an interpreted programming language and thus runs quite slowly. It does however provide a high level interface to the available graphics operations and interaction possibilities. This approach can be contrasted to the C++ programming language used in Zomit. C++ is compiled and fast. The developer of a virtual world in Zomit can use complex functions to generate the virtual world. This possibility is not available with Pad++. Animations that are not provided by the Tcl/Tk interface to the C++ substrate and that are too complex to be written in Tcl/Tk, can be implemented in Pad++'s KPL rendering language. This compiled language has a compact representation and executes about 100 times faster than Tcl.

Pad++ Has "Intelligent" Objects

Pad++ provides the graphical objects found in Zomit plus the inbuilt possibility to display Web pages in the hypertext markup language (HTML). Pad++ also provides input widgets, such as buttons and sliders, that are not available in Zomit. Graphical objects are created and positioned in the virtual world by Tcl code. This code specifies the size (in the virtual world as with Zomit), position, and colour. In contrast with Zomit, the notion of scale is not exposed to the developer. The visibility of objects is determined by their size. The developer specifies that an object

should only be visible when its size is within a given range. It is also possible to associate arbitrary Tcl code with events on objects. Code called when the mouse moves over an object can, for example, change the colour of the object. This code can also change the position of the user in the virtual world. These possibilities are not available in Zomit. In Zomit, semantic zooming is created by having many simple objects that are visible at different scales. In Pad++, code associated with objects is called when objects are rendered. This code has access to the current size in pixels of the object and can thus change the object's appearance depending on its size. Pad++ thus uses the capabilities provided by the interpreted language to create intelligent objects. These objects are drawn differently at different scales and thus remove the need, present in Zomit, for a different object to be created for each representation of the underlying data.

Optimisations

The implementation of Pad++ includes a number of algorithms designed to improve the efficiency of the display update code in order to achieve the highest frame rate possible. These algorithms include: spatial indexing of objects, index balancing, progressive display refinement, level-of-detail control, interruptible drawing tasks, and optimised image rendering. Most of these algorithms would be useful in Zomit. Other than some display optimisations (panning optimisations, asynchronous image scaling, and drawing already cached objects first), these algorithms have not yet been added to Zomit because the efficiency gains were not sufficiently important for the purposes of this project to justify the additional code complexity.

Conclusion

Pad++ is a now completed project for developing virtual worlds with intelligent objects. The implementation of these intelligent objects requires a “thick” client that has to be installed on each user's workstation.

6.4.4 Jazz's Architecture

Jazz is a ZUI written by some of the participants in the Pad++ project (Bederson et al., 2000). Jazz is a set of Java classes that uses the Java2D graphics library available in Java 2. The developer of the virtual world writes Java code to create instances of these classes. These instances can be visible components: rectangles, ellipses, text, images, polylines, polygons, shapes or encapsulated Java Swing widgets. Encapsulated widgets are scaled by remapping their input and output to take into account the transformation of their appearance. The other available

objects are called “nodes” and they allow the developer to place the visible objects into a 2D scene graph. The visual components specify what something looks like while nodes position visual components in the scene graph. This position describes when and where visual components are visible.

While visual components listed above have a fixed representation, nodes can be used to specify that they are visible only at certain magnifications. This mechanism is similar to that used in Zomit. The developer can, in addition to using the provided simple visual components, create custom components by extending a Jazz base class. In this case the rendering code in the visual component can directly adapt the appearance of the object to the current magnification.

The developer can also create lenses using a Jazz class that implements cameras. Cameras can be moved around the virtual world by the user. The rendering code in custom visual components can find out in what camera they are being drawn and change their representation in consequence.

The Jazz toolkit allows the user to visualize the virtual world described by the scene graph. The default event handlers in Jazz allow the user to pan with left mouse button, and control the scale with the right mouse button. The user presses on the right mouse button and drags to the right to zoom in, or drags to the left to zoom out. This interaction technique requires a mouse with at least two buttons and each action to be explained to users. The Control Menu used in Zomit only requires a single mouse button and once users have been told to press on a mouse button for the menu all the other commands are self-revealing.

The developers of Jazz chose not use 3D renderers such as OpenGL because they found that 3D renderers do not provide good support for 2D business graphics (scalable fonts, 2D complex polygons, line styles, for example) or standard user interface widgets (Bederson and Meyer, 1998). In addition, 3D renderers are optimised for convex 3D polygons (which 2D applications do not require).

Jazz provides a more reactive set of visible components than those available in Zomit. This was done by using a “thick” client (that takes longer to download), by using a more sophisticated graphics toolkit (Java 2 rather than Java 1) and by not having a client/server architecture (which means that the source of the virtual world cannot be too distant from the user).

6.5 Results and Perspectives

Zomit is a tool for creating Zoomable User Interfaces directly usable over the Internet. It is controlled by our new Control Menus that facilitate interaction with these complex interfaces and includes our new context aids which reduce user desorientation in large information spaces.

Zomit has been used to create two large virtual worlds. These are described

in the next chapter. Further work is need to improve the integration of Control Menus into ZUIs, in particular in an attempt to combine panning and zooming.

Chapter 7

Zomit Applications

Zomit is a development tool for Zoomable User Interfaces and as such has been used to create virtual worlds in two very different domains. We used Zomit to create an interface to a database containing genetic data and it has been used by another research laboratory to create an interface to a virtual library. These two virtual worlds can be tested with a standard Web browser at the Zomit home page: <http://www.infobiogen.fr/services/zomit/>.

It has also been used in student projects as a base for developing information exploration programs.

7.1 An Interface to a Genetic Database

As a demonstration of the use of Zomit, we developed a server, called ZoomMap, that queries the HuGeMap database of the major genetic and physical maps of the human genome. It provides a virtual world containing the human genome, chromosomes, maps, markers and sequences.

7.1.1 Why a Zoomable User Interface?

The volume of data produced in molecular biology has been growing exponentially for several years and all signs indicate that this process will continue during the next decade. This growth applies to mapping and sequence data on everything from microorganisms to humans.

For several years now it has been difficult for biologists to interact directly with the data concerning their species of particular interest. The huge volumes and the complexity of the data, and the numerous links between them, make it hard to maintain a global view of the data. Researchers are restricted to local views of their data, and long-distance relationships are not visible. Basically, there is

much more information available than researchers can interact with. Therefore, the selection of pertinent information is based on interactive, but rather arbitrary decisions, or on the results of an algorithm that processes the data according to a pre-established and rigid model.

With the large-scale sequencing projects now producing data at a rapid pace, the time has come to work on extracting knowledge from rich sources of data, i.e. to discover new models. Data mining can be used, but is not sufficient in the sense that it can only exploit the data to test hypotheses. Researchers need to interact globally with the data, to make incidental discoveries, find unexpected regularities and test new putative models.

The scientific community also suffers from a lack of interaction with the data, making the large-scale mapping and sequencing projects not as fruitful as they should be. For example, when using a World Wide Web (www) browsing tool applied to molecular biology databases, following a link from a page generates a view of the new page that is visually independent from the view of the first page. There is no visual track of the semantic relation between the two pages. Therefore users quickly forget the logic of their sequence of different views, and get lost. Often visualization software offers a view of the data according to a given perspective and modifying this perspective is not easy. These problems are general in most visualization and browsing software.

The technique of navigation by zooming is easily applied to molecular biology data for two main reasons. Firstly, the large number of links between objects in molecular biology leads to a complex graph of semantic relations which is best navigated by zooming (e.g. when focusing on a given node, the neighbouring nodes can be represented with edges decreasing with the minimal path length). Secondly, biological databases frequently contain a well-defined hierarchy of information, from a genome to its sequence for example.

7.1.2 HuGeMap Database

The HuGeMap database (Barillot et al., 1998) stores the major genetic and physical maps of the human genome. It includes:

1. the genetic maps from Généthon (Dib et al., 1996) and the Cooperative Human Linkage Consortium (Sheffield et al., 1995);
2. the physical maps from CEPH-Généthon (Bellanné-Chantelot et al., 1992; Cohen et al., 1993; Chumakov et al., 1995) and the Whitehead Institute-MIT (Hudson et al., 1995); and
3. the ISCN cytogenetic description.

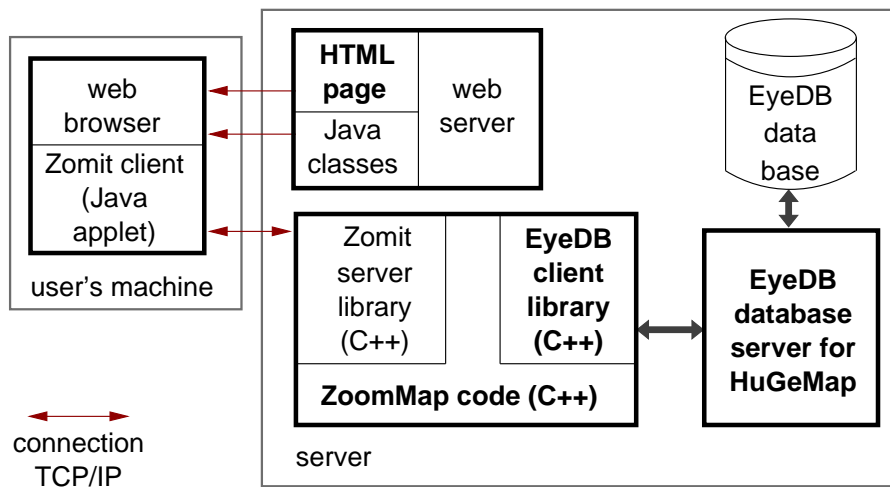


Figure 7.1: Components of ZoomMap

The schema of HuGeMap has a natural and strong hierarchy and contains a large number of links. On the top of the hierarchy is the human genome, then the chromosomes, their cytogenetic elements, the maps, the markers and finally the nucleotide sequences. Markers may belong to several maps. This database is, therefore, a good testbed for data visualization and browsing.

The HuGeMap database is stored in the object oriented database management system EyeDB (Viara et al., 1999) available at the URL <http://www.sysra.com/eyedb/>.

7.1.3 Using Zomit

ZoomMap uses the Zomit server library as shown in Figure 7.1. The ZoomMap code uses the EyeDB client library to read the HuGeMap database. It uses the data read from the database to create the objects that the Zomit server library send to the Zomit client. The ZoomMap server runs on the same machine as the EyeDB server. This allows ZoomMap and EyeDB to communicate via shared memory and to avoid the overhead of communicating via TCP/IP.

Reading the HuGeMap database is simplified by the fact that EyeDB is an object oriented database management system. In most cases, ZoomMap does not have to execute queries to retrieve the relevant information. The object oriented nature of the database means that ZoomMap just has to follow pointers to obtain the data. The simple fact of following the pointer causes EyeDB to read the required information from the database. This information is then immediately available in in-memory structures that can be directly used to generate the graphical objects from Zomit.

ZoomMap consists of about 2200 lines of C++ code.

7.1.4 ZoomMap: Zooming and Portals

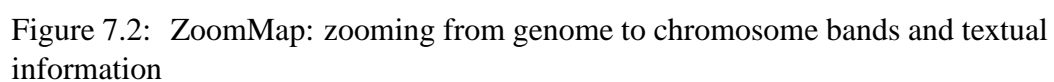
The initial view of the virtual world presented in ZoomMap consists of a karyotype of the human genome (Figure 7.2). The 24 chromosomes are depicted with their names and arms at the top level (Figure 7.2a). As the user zooms on a chromosome, its arms' names appear (Figure 7.2b), followed by the banding (Figure 7.2c). The names of the cytogenetic bands are shown when the bands are big enough to contain readable text. These steps are examples of semantic zooming (modification of the visualized object according to the scale). At the same time, more and more information on the chromosomes appears as text beside each chromosome. The names of the different maps associated with each chromosome are displayed and, if one continues to zoom, the maps themselves are drawn (Figure 7.3a) using their usual representation (markers positioned on an axis). Only those markers for which space is available are shown, another example of semantic zooming.

When there is sufficient space a series of portals are shown opposite each marker (Figure 7.3b). They point to the other maps in which the corresponding marker is present. After further zooming on a marker, the sequence appears under the name (Figure 7.3c).

7.1.5 ZoomMap's Magic Lenses

Several different types of Magic Lenses have been implemented (Figure 7.4).

- Lenses that display a map when applied to a cytogenetic description of a chromosome (Figure 7.4a and Figure 7.4b). This representation of a chromosome is also given when zooming in further from the chromosome name. This illustrates the fact that a problem of data transformation can be addressed with different techniques: here Magic Lenses or semantic zooming. Lenses present the advantage of offering numerous possibilities simultaneously (one can have as many lenses as necessary and choose among them) while with semantic zooming there is only one possible transformation.
- A lens that, when applied to markers, uses the colour of each marker to show the level of heterozygosity (Figure 7.4c). Here, a marker's heterozygosity is coded on a scale from blue (100% heterozygous) to white (0% heterozygous).
- A lens that displays only those markers with high heterozygosity (>65%; Figure 7.4d). Here the lens is used as a filter to select some data. Typically,



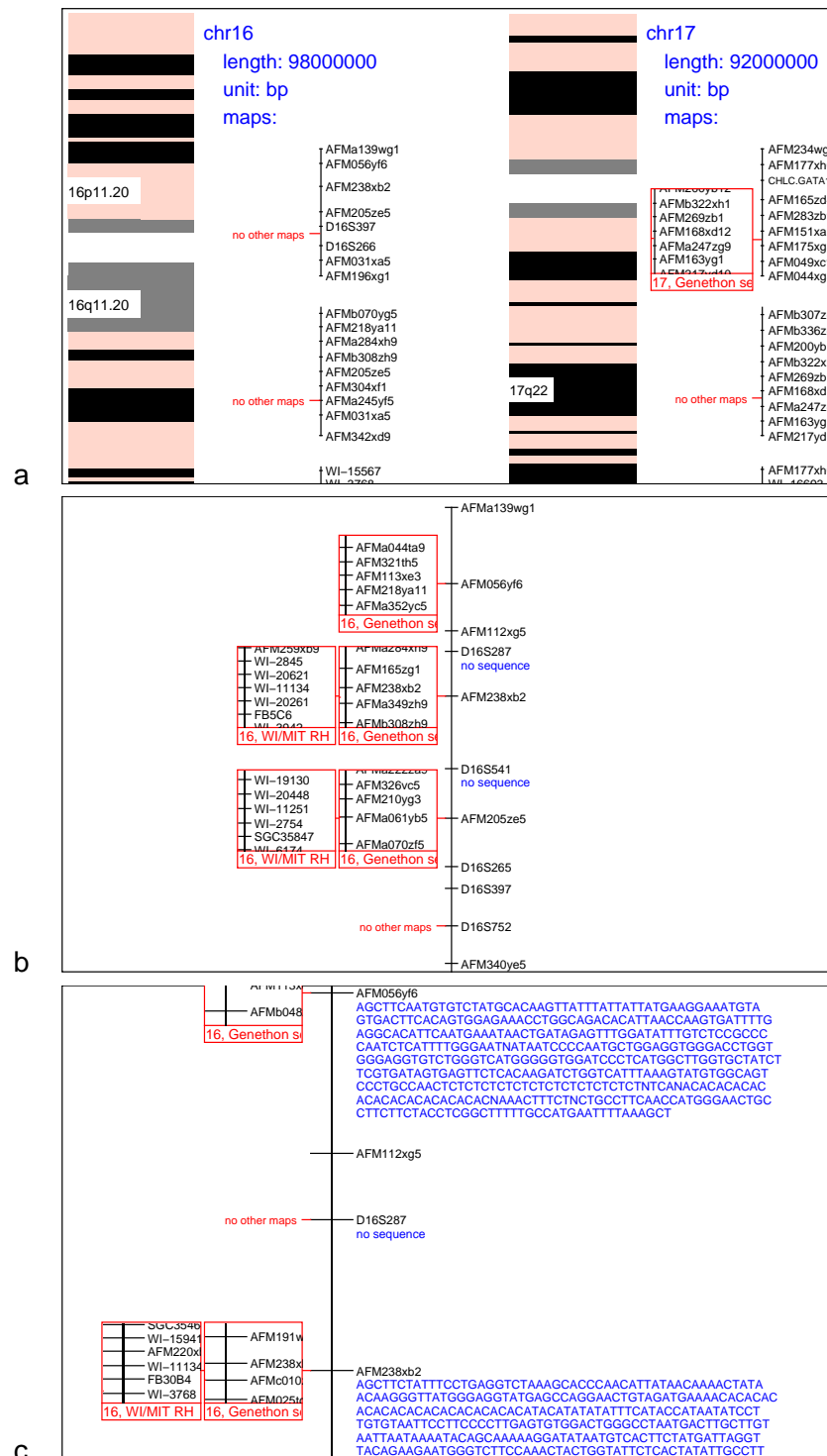


Figure 7.3: ZoomMap: zooming from maps to sequences

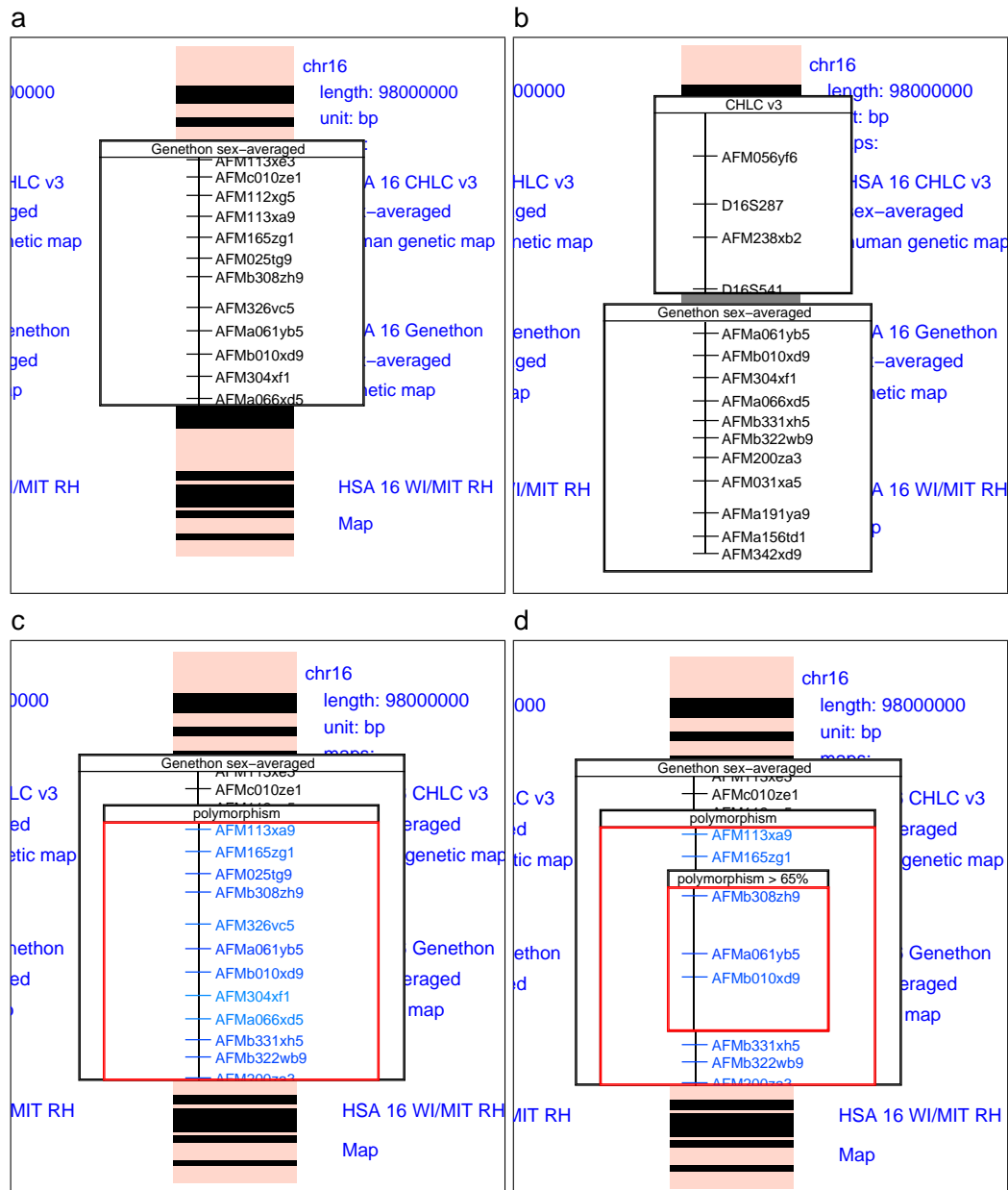


Figure 7.4: ZoomMap: (a) transformation of the cytogenetic representation of a chromosome into its Génethon genetic map; (b) as for (a) with two lenses; (c) combination of the first lens and a lens that encodes the marker heterozygosity on a blue scale; (d) as for (c) with a third lens that selects only the markers with heterozygosity greater than 65% (the marker AFM326cv5 is no longer shown).

such lenses can be combined to produce new filters on a conjunctive (logical AND) basis.

- A lens that gives further information on each marker: name, D-number, EMBL access number, and heterozygosity. This lens is a magnifying glass in order to have sufficient space to display the additional information (Figure 4.36).

As discussed in subsection 4.3.6, these lenses can be combined as shown in Figure 7.4c and Figure 7.4d.

7.1.6 Zooming On Indices

Our purpose is to visualize biological data, generally stored in structured databases, for which indices exist. (Indices are sorted lists of object identifiers.) Though the logic of navigation by zooming is based on an hierarchically organised data schema, it may be useful to browse through these indices to retrieve data.

In ZoomMap the user can zoom on three different indices (D-number, EMBL access number, and name) that are present on the right-hand side of the world at the beginning of a session (Figure 7.5).

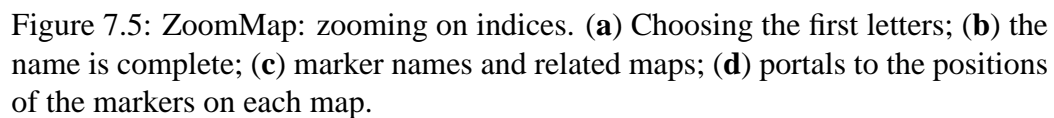
In the initial view, all the possible first letters of the indexed fields are displayed in order and with a size reflecting the number of objects starting with each letter. After zooming in on the desired first letter, the second and following letters appear as soon as space is available, or a range of strings if all the possibilities would span too much space (Figure 7.5a). Different colours encode the type of the string shown; i.e. the current position in the index, string ranges, and complete marker names.

This repeated zooming process leads to the complete name of the desired object (Figure 7.5b and Figure 7.5c), and then to a series of portals that point towards the position of the marker in the maps in which they are present (Figure 7.5d).

This example of ZoomIndex is trivial but one can imagine its use for more sophisticated purposes; a two-dimensional ZoomIndex would for example cross two indexed fields and could be used for interactive optimisation.

7.1.7 Evaluation

To aid the evaluation of the visualization techniques proposed in this paper we created a modified version of the ZUI without the hierarchy trees. Eight subjects, chosen from our colleagues at Infobiogen, were taught how to use our ZUI. The subjects were asked to answer 22 multiple choice questions. Some of the questions were (translated from French):



1. What is the name of the pink sub-band above the centromere of the chromosome 19?
☐ 19p13.13
☐ 19p13.10
☐ 19p13.11
☐ 19p12.30
2. How many sub-bands are under the band 9q34?
☐ 1, ☐ 2, ☐ 3, ☐ 4, ☐ 5
3. What are the names of the last marker of the map CHLC of the chromosomes 15, 10, & 5?
15 ☐ CHLC.GATA27A03
☐ AFM072yb11
☐ D15S641
☐ AFM262xb1

A training session explained how to answer these question with and without the hierarchy trees. This experimental design allowed us to study, using two interfaces otherwise as similar as possible, whether the hierarchy trees were of assistance.

Ordering effects were taken into account. Half of the subjects answered their first 11 questions with the hierarchy trees and the other half of the subjects answered their first 11 questions without the hierarchy trees. Half of the subjects were given the first 11 questions in the list of 22 to do first while the other half of the subjects did the second half of the questions first. This lead to four equally sized groups of subjects.

For each subject we calculated the time taken to answer 11 questions without the hierarchy trees divided by the time taken to answer the other 11 questions with the hierarchy trees. A value greater than one from this calculation would mean that having the context aids was an advantage. The mean value was 1.58 with a standard deviation of 0.54. The high standard deviation was caused by the lack of familiarity of some of the subjects with ZUIs. For these people the training session was not long enough and they thus found the second set of questions easier. In general however the subjects were faster with the hierarchy trees and were positive in their comments regarding these aids: in fact those that started with the hierarchy trees were reluctant to continue the experiment without them.

7.2 CDI: a Zoomable Virtual Library

The École des Mines de Nantes used Zomit to develop an interface for a virtual library. This virtual library contains 3000 books and is very similar to part of a real library in a department of this university. This ZUI, called CDI, was developed

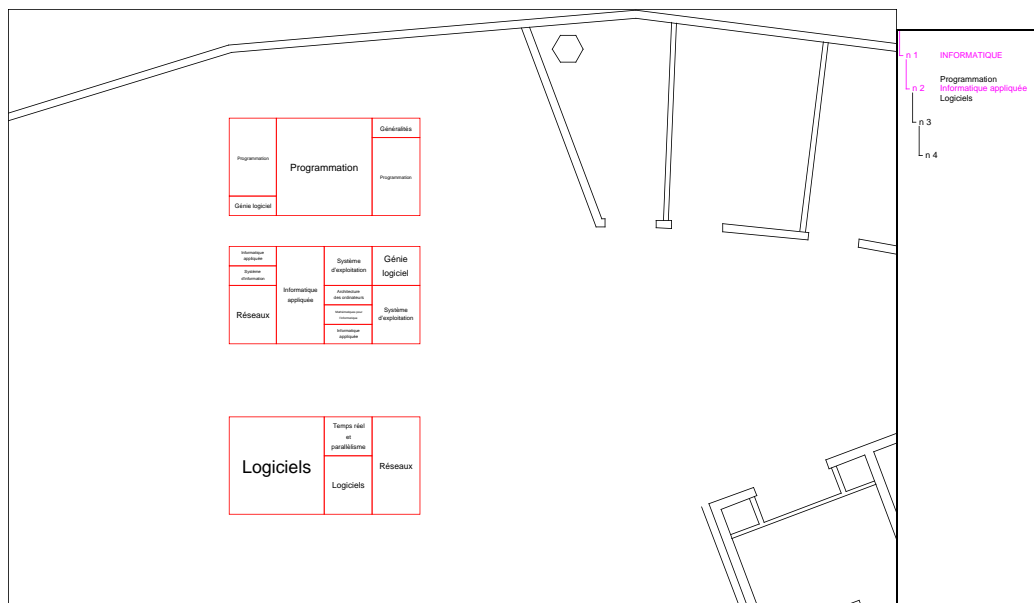


Figure 7.6: Global view of the library's shelves

as part of a project to analyse the time taken to find books using three different types of interface to the virtual library (Lecolinet et al., 2001). The two other interfaces were a three dimensional virtual reality interface and a two dimensional tree browser similar to those discussed in section 4.2. The speeds in finding book in these three interfaces are to be compared to those in a real library. The study will attempt to understand the advantages and disadvantages of the different representations and quantify any transfer of learning from the virtual interfaces to the real world and vice versa.

With this ZUI users can zoom from a global view of the shelves (Figure 7.6) to a view of the different subject treated in the library (Figure 7.7). Once users have identified which shelf is likely to contain the required book they can continue zooming to images of the books' covers (Figure 7.8).

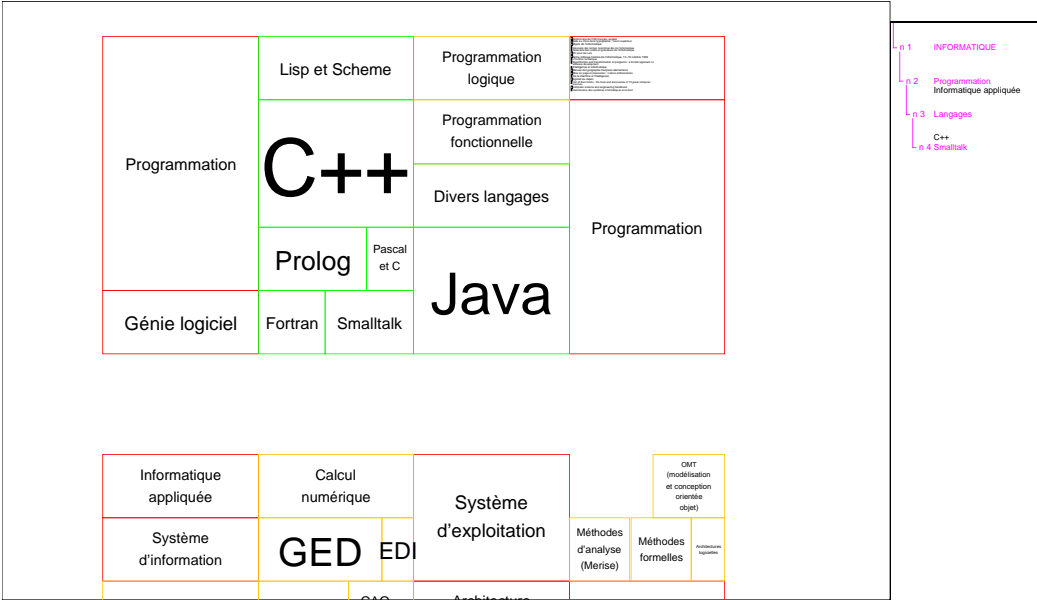


Figure 7.7: One of the library's shelves

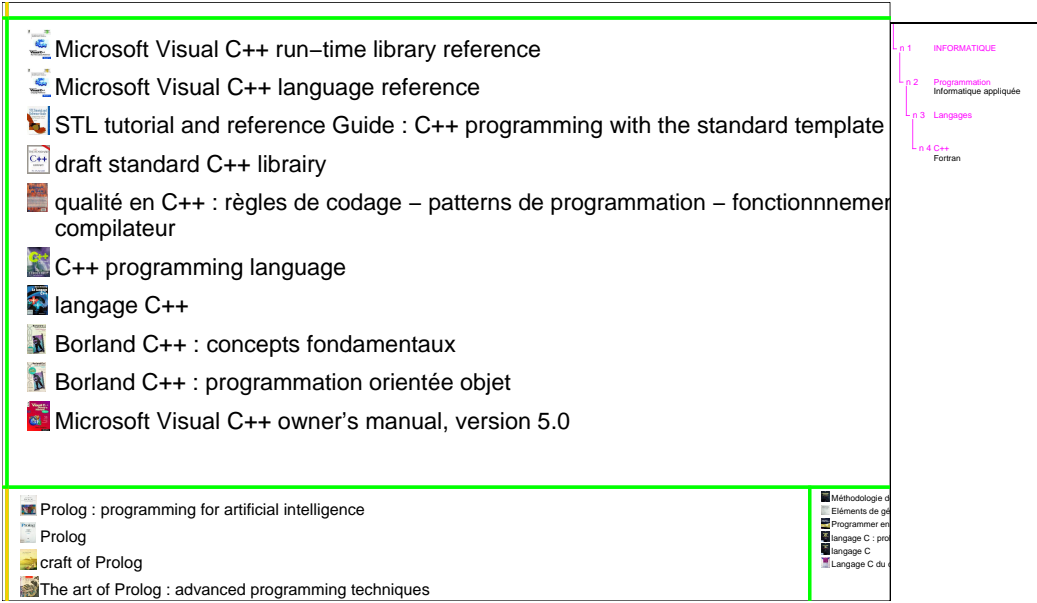


Figure 7.8: Some of images of the library's books

Chapter 8

Conclusion and Perspectives

The visualization of databases is an important part of many tasks in research and industry. This is summarised in the phrase: “seeing helps thought”. Zoomable User Interfaces are one tool used to create virtual worlds for large information spaces and to allow users to navigate in these worlds. The aim of this navigation is to help users find the information that they are looking for and allow them to transform this information into the required representation. Even with these tools finding the required information is frequently difficult. The information spaces are very large and the lack of contextual clues often means that users become disoriented before they find what they are looking for or once they have found it they are unable to relate it to the information space in general.

We have proposed three new contextual aids that help users understand the information space, their position in that space, and the relationships between the objects currently visible and the information space in general. The first of these aids is a permanent view of the hierarchy present in many information spaces. This view shows users where information is in the virtual world and the current position of the user. The second aid is a transient transparent view that users create when necessary and which vanishes automatically when no longer needed. It shows the current view in relation to more global views. The global view is chosen by the user with a dynamic and continuous user control. This aid thus helps disoriented users reposition themselves with respect to whatever global or context views are most useful. The third aid is also transient and transparent. It is created by users when they need to find out how they arrived in their current position in the information space and desire to see this path in relation to their current position and a global view of the space.

All interfaces to large information spaces have the problem that there are very large quantities of information to display on relatively small screens. Also, in order to understand relationships between different data items, users need to ap-

proach items from widely separate areas in the information space. It is not sufficient to see information in only one representation; understanding data often requires it to be seen in a number of different representations. The first of the two standard solutions is to show many different views in adjacent windows, called space multiplexing. The problem is that there is often not enough screen space and, even if there is, the focus changes between separate windows are a hindrance to the integration of the views into a single mental picture of the data. The second solution is to show the views at different times, but then users have to memorise one view in order to compare it with the next. Magic Lenses use this time multiplexing technique as they cover the focus of the user's attention in order to show this area with a different representation.

We have attempted to solve this problem with the use of transparent layers that can show two types of information in the same screen space. Users can use depth information or, in our system, movement caused by the user's control of the interface to separate the two views. We discovered that a fluid and continuous control leads to movement between the views which then makes it easier to distinguish one view from another. Our control mechanism makes it possible to view two types of information in the same screen area. This both conserves screen space and helps users integrate the two views by avoiding the changes of focus of attention between two separate views.

The results presented here are preliminary. They need to be validated by controlled experiments that compare the techniques presented in this thesis with the visualization systems discussed in chapter 4. Only such experiments will show in which situations the new techniques are superior to those used in existing visualization systems.

Our contextual aids rely on a synergy between interaction and visualization for their ease of use and effectiveness. To supply this synergy, missing from current interactors, we proposed a new type of pop-up menu, a Control Menu, that allows users to select operations from a menu and to then control the operation. This selection and control is performed in a single gesture that creates a bonding between menu use and execution. It also avoids the use of multiple interactors, a menu then a dialog box containing sliders and a button, to perform what is conceptually a single action. Control Menus have a fluidity of execution and feedback that ensures that users have and see exactly the result that they want before they end their gesture. Once again we reduce the number of changes of focus of attention, this time by avoiding the use of several interactors for a single task.

We have concentrated on the use of Control Menus in Zoomable User Interfaces with some tests of their use in virtual reality environments. We consider however that these menus can be useful in many different types of applications and that further work, and controlled user experiments, should be conducted in this area.

This thesis has shown that the creators of visualization systems must design the visualization techniques and the interaction aspects of their systems conjointly. What visualization systems can show to users depends on how users control these systems. Immediate visual feedback is important in helping users understand interactions and allows them to use successive refinement as they move towards the desired result.

Chapitre 9

Synthèse

9.1 Introduction

Les bases de données utilisées pour la recherche et dans les entreprises ont grandi de façon exponentielle pendant de nombreuses années, et tout laisse à penser que cette croissance continuera. Il ne suffit pas de stocker des données dans une base pour comprendre l'information qu'elles contiennent. Pour convertir ces données en connaissances, les utilisateurs doivent pouvoir interagir avec les bases de données.

La grande quantité de données, leur complexité, leur diversité, et le grand nombre de liens entre ces données poussent les utilisateurs à se servir de systèmes de visualisation sophistiqués. Ces systèmes doivent leur permettre d'obtenir une compréhension globale d'un espace d'information et de diriger leur attention sur la région de l'espace qui est intéressante pour une tâche donnée. Cependant, même quand un utilisateur est en train de se concentrer sur une petite partie d'un espace de grande taille, il doit pouvoir rester conscient de sa localisation par rapport à l'espace entier. Ceci doit permettre une meilleure compréhension des relations entre les données visibles et celles situées dans d'autres régions de l'espace.

9.1.1 Amélioration de l'interaction

L'utilisation effective de techniques d'interaction complexes nécessite un contrôle continu de l'interface par ces utilisateurs. Ceux-ci doivent spécifier quelle partie de l'espace doit être visible et comment cette région doit être présentée. Mais ces choix ne sont pas fixes car l'utilisateur change la région d'intérêt et la façon de la présenter très fréquemment. Le contrôle de ces choix doit être fluide et prendre en compte le fait que l'interface utilise changement et mouvement comme moyens de visualisation autant qu'elle utilise des présentations fixes.

Les interacteurs standard n'ont pas beaucoup évolué depuis le développement du modèle d'interaction WIMP (Window, Icon, Menu and Pointer device) et l'invention de la manipulation directe. Ces interacteurs ont été prévus pour effectuer des suites de modifications distinctes sur une interface qui est essentiellement statique. Ils ne sont donc pas adaptés au contrôle fréquent ou continu d'une interface changeante.

De nombreux interacteurs permettent aux utilisateurs de choisir une opération parmi une liste d'opérations. Ces interacteurs sont souvent des menus dont les plus récents ont été conçus de telle sorte que des utilisateurs novices et les utilisateurs experts les utilisent presque de la même façon (ce qui aide les utilisateurs novices à devenir « experts »).

D'autres types d'interacteurs permettent aux utilisateurs de contrôler interactivement l'opération choisie. Ces outils, tels que les barres de défilement et les boîtes de dialogue, sont utilisés après la sélection de l'opération mais sont indépendants de l'interacteur de sélection. Cette séparation de la sélection des opérations et de leur contrôle force les utilisateurs à partager leur attention entre deux interacteurs différents : l'un dédié à la sélection, l'autre au contrôle de l'opération.

Dans la première partie de cette thèse, nous présentons et discutons les différents menus utilisés actuellement. Nous présentons ensuite un nouveau type de menu, appelé *Control Menu*, qui combine la sélection des commandes et leur contrôle en un seul interacteur. Cet interacteur n'utilise que la souris et un seul de ses boutons. Cet interacteur intégré fournit un contrôle rapide et fluide pour des systèmes complexes de visualisation.

9.1.2 Davantage de contexte

De nombreuses techniques de visualisation ont été proposées dans les communications de recherche et certaines de ces techniques ont été utilisées avec succès dans des logiciels commerciaux. Notre travail concerne plus particulièrement les systèmes de visualisation qui montrent une vue de l'espace d'information avec laquelle les utilisateurs peuvent interagir pour trouver les informations recherchées. Les systèmes de ce type, dont un certain nombre sont présentés dans la deuxième partie de cette thèse, peuvent être classifiés selon différentes taxonomies. Nous présentons d'abord quelques taxonomies proposées par d'autres auteurs puis une nouvelle taxonomie. Tous ces systèmes sont confrontés au même problème : comment afficher le contexte des vues de détail. Les utilisateurs deviennent vite désorientés si la quantité de contexte visible n'est pas suffisante, même après une courte durée de navigation : ils ne savent plus se localiser dans l'espace d'information ni où trouver les informations recherchées. Ils sont « perdus dans l'hyperespace ».

Les interfaces zoomables (« zoomable user interfaces » ou ZUIs) sont un type

de systèmes de visualisation prometteur pour représenter de grands espaces d'information. Les ZUIs sont basées sur le concept de zoom sémantique. Elles permettent de créer un monde virtuel multidimensionnel où les utilisateurs peuvent trouver et transformer l'information d'une base de données. L'interaction avec ces interfaces implique une utilisation fréquente de certaines commandes. Les deux plus importantes sont « zoomer » (agrandir) et faire défiler la vue (se déplacer dans l'espace). Nous avons développé une ZUI qui utilise notre Control Menu et intègre donc la sélection et le contrôle de ces commandes en un même geste.

Les ZUIs souffrent souvent d'un manque de contexte. Après avoir navigué dans l'espace d'information (et donc après avoir quitté la vue globale initiale de l'espace) les utilisateurs voient une vue très partielle du monde virtuel qui contient peu d'information contextuelle. Ceci aggrave les risques de désorientation.

Nous proposons un multiplexage spatial et en profondeur pour ajouter du contexte aux interfaces zoomables. Le multiplexage de l'espace permet d'afficher les mêmes informations avec des représentations différentes au même moment. Notre ZUI montre une vue zoomable de l'espace d'information et une représentation hiérarchique du même espace dans deux fenêtres synchronisées. La représentation hiérarchique montre une vue en coupe de la structure de l'espace d'information et le couplage avec la vue zoomable indique aux utilisateurs leur position pendant la navigation.

Nous proposons également deux autres aides contextuelles qui utilisent des vues transparentes avec multiplexage en profondeur et qui sont contrôlées par notre nouvel interacteur, un Control Menu. Ces aides sont temporaires et créées à la demande de l'utilisateur. Elles n'existent que pendant le geste qui les a créées et qui les contrôle. Leur utilisation demande un contrôle fluide et continu car leur utilité vient de leur réactivité. Ce contrôle est fourni par des Control Menus. Le mouvement qui résulte de ce contrôle continu aide les utilisateurs à séparer l'aide transparente de la vue, toujours visible, du focus.

Après une présentation et une classification des systèmes existants, la deuxième partie de cette thèse présente nos nouvelles aides contextuelles pour les ZUIs puis Zomit, un outil de développement qui facilite la mise en œuvre des interfaces zoomables. Cette partie se termine par une présentation des applications créées avec cet outil.

9.2 Interaction

Cette section examine des menus actuellement utilisés pour contrôler des systèmes informatiques, les inconvénients de ces menus, et comment un nouveau type de menu peut pallier ces inconvénients.

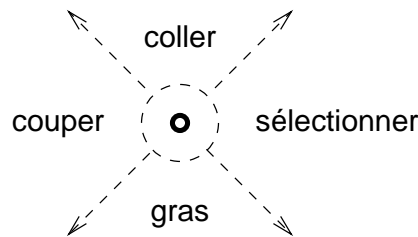


FIG. 9.1 – Un pie menu et, en pointillé, ses zones actives

9.2.1 Menus classiques et contrôle des opérations

Typiquement, l'interaction via une interface utilisateur comprend la sélection d'un (ou plusieurs) objet(s), la sélection d'une opération puis le contrôle de l'exécution de cette dernière. Dans la plupart des interfaces, la dernière phase de cette interaction est séparée des premières phases bien qu'elles soient en fait vues comme un tout par l'utilisateur. Ces deux parties de l'interaction sont généralement effectuées avec des interacteurs différents qui s'utilisent de façons distinctes. Cette complexité supplémentaire ralentit l'interaction car des actions simples doivent être exécutées en deux étapes. En effet, l'utilisation de plusieurs interacteurs provoque une interaction moins fluide du fait des multiples changements du focus d'attention de l'utilisateur. De plus, si l'utilisateur doit utiliser encore un autre interacteur pour sélectionner, l'interaction est encore plus ralentie.

Dans cette sous-section, nous présentons une analyse des techniques de sélection et de contrôle, et les problèmes venant de l'utilisation d'interacteurs multiples pour effectuer une seule opération.

Menus existants

Des menus standard tels que les menus déroulants et les menus contextuels permettent aux utilisateurs de choisir facilement des opérations à effectuer. Les menus contextuels sont activés à un endroit choisi par l'utilisateur dans l'interface. L'interface peut alors adapter le contenu du menu à la position choisie et associer l'action choisie à l'objet se trouvant à cette même position. Par ailleurs, de nouveaux menus contextuels, les « pie menus » (Callahan et al., 1988; Hopkins, 1991) et les « marking menus » (Kurtenbach and Buxton, 1994), ont été proposés afin de rendre l'utilisation des menus contextuels plus rapide.

La différence la plus importante entre un pie menu et un menu contextuel standard est le fait que les entrées dans un pie menu sont distribuées autour du centre du menu (Figure 9.1). L'utilisateur n'a plus besoin de sélectionner une ligne dans une liste linéaire mais peut tout simplement déplacer le pointeur dans la bonne

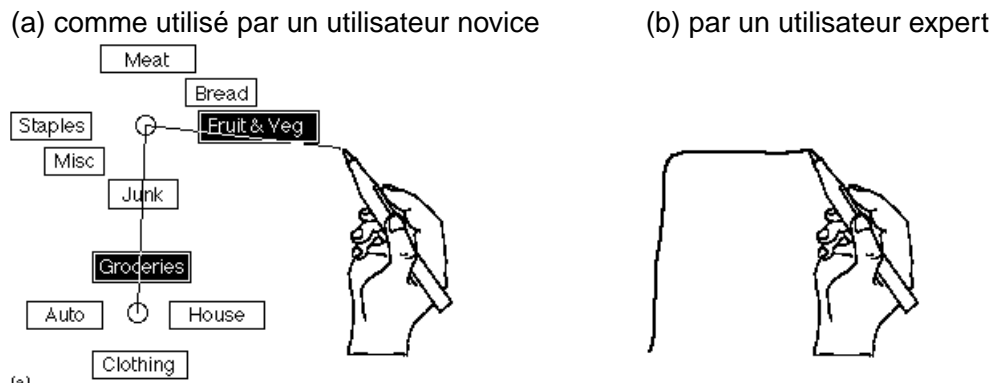


FIG. 9.2 – L'utilisation d'un marking menu (copié de Kurtenbach and Buxton, 1993)

direction et relâcher le bouton de la souris. Un utilisateur expert n'a même plus besoin de voir le menu pour choisir la bonne opération et dans ce cas-ci le menu ne s'affiche pas. Les pie menus sont par exemple utilisés dans le jeu « The Sims » (<http://www.thesims.com>) pour contrôler rapidement les actions des personnages. Ces menus contextuels permettent d'associer des actions aux personnages sans sélectionner préalablement le personnage qui doit être le sujet de l'action. Ainsi, un joueur expert peut faire des actions rapidement car il n'est pas obligé d'attendre que le menu s'affiche pour y sélectionner une action. Un marking menu ressemble à un pie menu sauf que chaque fois que l'utilisateur active une opération, le système dessine un trait sur l'écran indiquant le geste optimal que l'utilisateur aurait pu faire. Un marking menu peut également avoir des sous-menus. La Figure 9.2 montre un marking menu avec un sous-menu dans les cas d'utilisation novice (à gauche) et experte (à droite). Dans les deux cas le système dessine le trait du geste optimal.

Analyse de menus

La loi de Fitts (MacKenzie, 1995; Raskin, 2000) estime le temps nécessaire pour déplacer le curseur vers un bouton sur l'écran. Ce temps est fonction de la taille du bouton et la distance entre le curseur et ce bouton selon la formule $t = a + b \log_2(d/s + 1)$ où s est la taille du bouton et d la distance entre le curseur et le bouton. Cette loi dit donc que les menus contextuels devraient être rapides à utiliser car l'utilisateur n'est pas obligé de se déplacer pour se servir du menu. D'après cette même loi, les pie et marking menus sont encore plus avantageux car, pour choisir une opération, il suffit de déplacer le curseur vers une des zones actives du menu (indiquées en pointillé dans la Figure 9.1). Ces zones sont grandes et proches du curseur (le menu s'affiche de sorte que le centre du menu soit sous

le pointeur). Rapidement l'utilisateur apprend la position des opérations les plus fréquemment utilisées et peut alors se servir du menu sans avoir à le regarder. Cette facilité de sélection contraste avec le temps pris pour sélectionner la bonne ligne dans un menu contextuel de type linéaire.

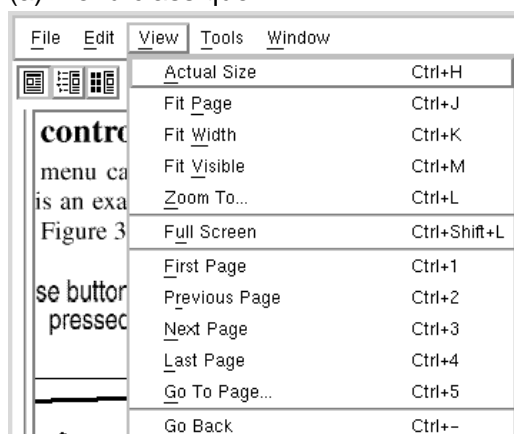
Opérations avec des paramètres continus

Une opération continue est une opération comportant au moins un paramètre de type numérique (avec un nombre important de valeurs possibles). La sélection de l'échelle d'une ZUI est un exemple d'opération continue. Les menus contextuels (les pie et marking menus ainsi que les menus contextuels standard) ne permettent pas aux utilisateurs de contrôler l'opération choisie de façon continue (par exemple pour effectuer un défilement ou zoomer jusqu'à ce que l'utilisateur trouve la bonne taille). Ils ne permettent pas non plus aux utilisateurs de fournir des paramètres pour contrôler l'opération sélectionnée. Par exemple, une opération telle que le changement de la taille de police dans un traitement de texte nécessite souvent l'ouverture d'une boîte de dialogue pour entrer un paramètre : la nouvelle taille. Les utilisateurs doivent d'abord ouvrir le menu et sélectionner la bonne opération puis se concentrer ensuite sur un deuxième interacteur, typiquement une boîte de dialogue. Une fois la nouvelle taille entrée, la boîte de dialogue disparaît et l'utilisateur doit à nouveau changer de contexte. Si la taille choisie n'est pas bonne, l'utilisateur doit alors recommencer. Cette interaction nécessite plusieurs changement de focus et des déplacements de souris qui ralentissent l'interaction.

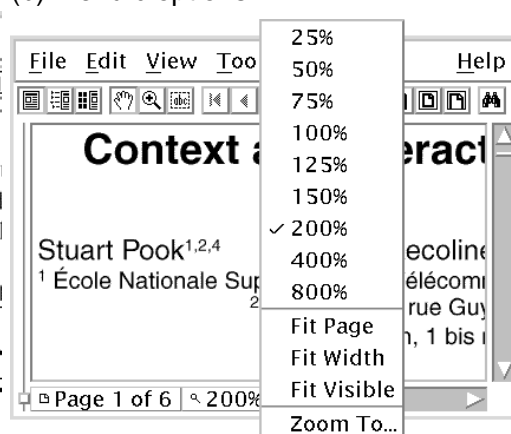
Un exemple : l'échelle d'Acrobat Reader

Le contrôle des opérations continues telles que zoomer et défiler dans des applications comme Acrobat Reader d'Adobe est un exemple de l'insuffisance des interacteurs existants. Cette application propose quatre façons différentes de zoomer : via un menu classique (Figure 9.3a), via un « option menu » (Figure 9.3b), via une boîte de dialogue (Figure 9.3c), ou en cliquant avec la souris mais après avoir sélectionné un mode *ad hoc* (Figure 9.3c). L'utilisation d'un « option menu » ou d'une boîte de dialogue provoque les problèmes d'interaction précédemment cités. De plus l'« option menu » occupe une place non négligeable sur l'écran. Ces deux interacteurs ne sont pas contextuels et ne permettent donc pas à l'utilisateur d'indiquer sur quelle région de l'écran il veut centrer le zoom. Il sera donc nécessaire de recentrer la vue affichée après chaque zoom. Dans le quatrième cas, l'utilisation de la souris pour zoomer minimise les changements de point d'attention et évite d'avoir à recentrer la vue mais nécessite par contre d'avoir préalablement sélectionné un mode spécifique (car le bouton de la souris sert également

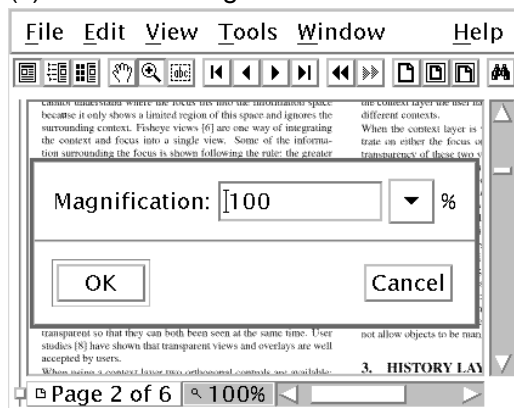
(a) menu classique



(b) menu d'options



(c) boîte de dialogue



(d) souris

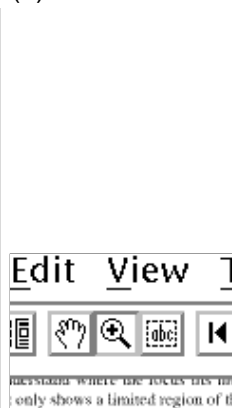


FIG. 9.3 – Modifier l'échelle dans Acrobat Reader 3

à d'autres types d'actions dans d'autres modes). De plus, ces trois interacteurs changent l'échelle par incréments pré-définis et l'utilisateur devra donc les activer plusieurs fois avant d'obtenir l'échelle désirée. Les mêmes problèmes se posent pour faire défiler le document à l'aide de barres de défilement qui utilisent encore plus de place que l'« option menu » (elles fournissent néanmoins une information sur la position de la vue courante dans le document).

9.2.2 Un nouvel interacteur

Cette section présente un nouveau type de menu contextuel, nommé « Control Menu ». Ce nouvel interacteur est particulièrement bien adapté pour contrôler des interfaces complexes comme les ZUIs. Son usage reste cependant tout à fait général et n'est pas limité à ce type d'interface (comme nous le verrons dans la section suivante).

« Control Menu »

Nous proposons un nouveau type de menu contextuel qui permet à l'utilisateur de fournir jusqu'à deux paramètres à l'opération choisie ou de la contrôler dans une ou deux dimensions indépendantes. Ce menu permet ainsi aux utilisateurs de *choisir* et de *contrôler* des opérations en un seul geste.

Un Control Menu a des similarités de comportement avec un pie menu. Un utilisateur, qui ne sait pas où se trouve l'opération qu'il désire, enfonce le bouton de la souris, attend 0,3 secondes (temps expérimental proposé par (Kurtenbach and Buxton, 1994)) jusqu'à ce que le menu soit affiché centré sous le curseur, puis déplace ce dernier dans la direction de l'opération désirée (Figure 9.4). Le menu disparaît et l'opération commence dès que le curseur a été déplacé de la *distance d'activation* depuis le centre du menu (nous avons empiriquement choisi une distance d'activation de cinq fois le rayon du cercle au centre du menu). L'opération se termine quand l'utilisateur relâche le bouton de la souris. Les mouvements de la souris effectués pendant l'opération fournissent les paramètres nécessaires au contrôle de cette opération. Un utilisateur qui sait où se trouve la commande souhaitée fait le même geste qu'un novice mais sans effectuer la pause qui fait apparaître le menu. Ainsi, les utilisateurs experts ne sont pas distraits par l'apparition du menu et les utilisateurs novices apprennent progressivement le geste expert.

Un Control Menu comprend jusqu'à huit flèches et un libellé par flèche. Les libellés sont désignés en caractères noir sur fond blanc (ou du caractères gris pour les choix inapplicables à ce moment). Outre ces éléments, le menu est transparent afin de ne pas cacher la zone d'intérêt lorsque l'utilisateur est en train de choisir l'opération.

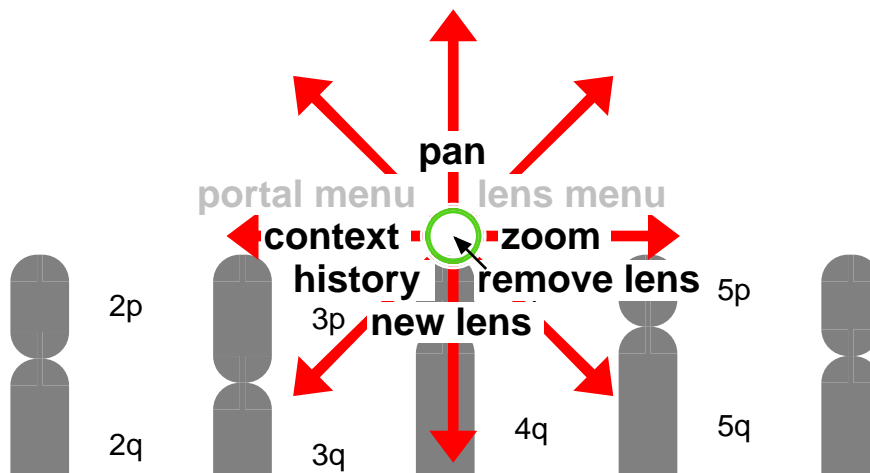


FIG. 9.4 – Un Control Menu (sur la vue de la Figure 9.16a)

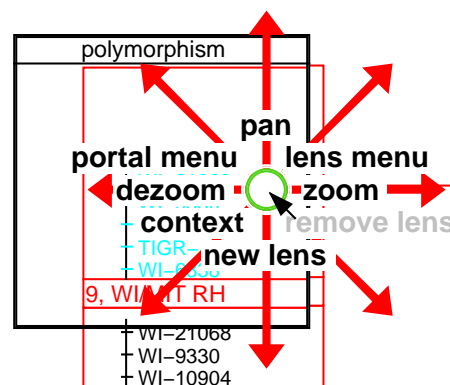


FIG. 9.5 – Un Control Menu sur deux types d'objet

Le contenu d'un Control Menu peut s'adapter à la position où l'utiliser le fait afficher ou aux objets se trouvant à cette position. Ceci peut éviter de proposer des opérations qui ne sont pas appropriés à un moment donné. La Figure 9.5 montre le même menu que dans la Figure 9.4 sauf que davantage d'options sont disponibles (car le menu se trouve sur un portail et sur une lentille). Dans ce cas c'est le choix de l'opération qui déterminera quel objet sera le sujet de l'opération.

Opérations contrôlées par un seul paramètre

Une entrée dans un Control Menu peut par exemple servir à modifier le niveau de zoom dans une ZUI. Cette opération illustre l'intégration d'un menu et d'une barre de défilement dans un seul interacteur. La Figure 9.6 montre les mouvements de la souris pendant l'utilisation d'un Control Menu pour choisir et contrôler une

opération de zoom ou de dézoom. L'utilisateur enfonce le bouton de la souris et déplace celle-ci de la distance d'activation (le mouvement numéro 1 dans la Figure 9.6) vers la droite (l'opération « zoom » étant sur la droite du Control Menu représenté à la Figure 9.4). Ceci amorce l'opération de zoom et la forme du pointeur change à ce moment sur l'écran. À partir de ce moment, les mouvements de la souris vers la droite (mouvements 2 et 4 dans la Figure 9.6) zooment la vue et les mouvements vers la gauche (mouvement 3) la dézooment. Le contrôle par retour d'information est immédiat : la vue change dès que l'utilisateur bouge la souris. L'utilisateur relâche le bouton de la souris lorsque l'échelle voulue (un dézoom dans la Figure 9.6) a été obtenue. Cet exemple montre qu'avec le menu de la Figure 9.4, pour dézoomer il faut d'abord zoomer un peu. Alternativement, une entrée « dézoomer » pourrait être ajoutée dans la partie gauche du menu. L'utilisateur déplacerait alors directement la souris vers la gauche pour dézoomer. Cette solution a cependant le désavantage d'utiliser davantage de place dans le menu et sa mise en œuvre est donc dépendante des spécificités de l'application.

Pendant l'opération de zoom, l'utilisateur peut annuler cette opération en déplaçant, sur une grande distance, la souris dans la direction orthogonale (vers le haut ou vers le bas dans le cas de l'exemple). L'utilisateur peut alors confirmer l'annulation en relâchant le bouton de la souris ou infirmer cette annulation en retournant la souris vers sa position verticale initiale. Dans ce cas, le bouton de la souris reste enfoncé et l'utilisateur peut continuer de zoomer. Il n'est pas possible d'annuler toutes les opérations de cette manière ; seules les opérations où une seule des deux directions de déplacement de la souris est utilisée peuvent être annulées de cette manière.

Opérations contrôlées par deux paramètres

Un Control Menu peut aussi être utilisé pour effectuer des défilements bi-directionnels. Il remplace alors deux barres de défilement. L'opération de défilement est sélectionnée en enfonçant le bouton de la souris et en déplaçant la souris vers le haut (« pan » sur la Figure 9.4 en haut du menu). La vue suit le curseur

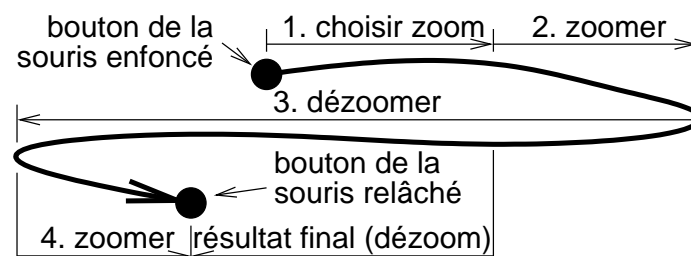


FIG. 9.6 – Zoomer avec un Control Menu

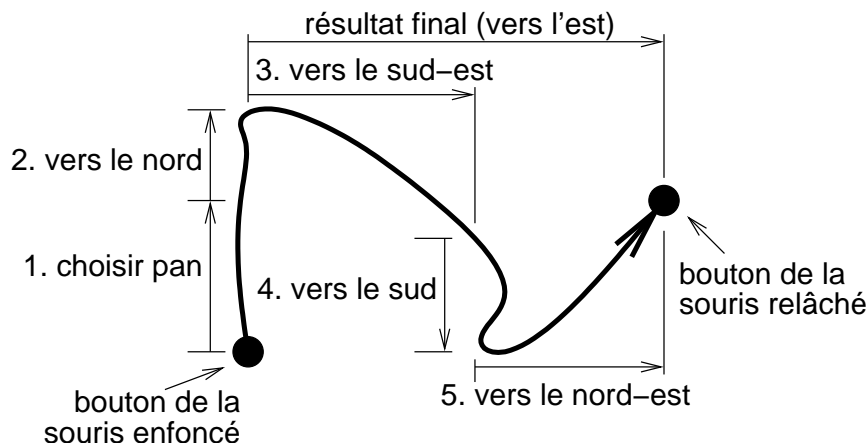


FIG. 9.7 – Défilement avec un Control Menu

pendant l'opération (Figure 9.7). Cette opération ne peut pas être annulée pendant l'opération car les deux sens de déplacement de la souris ont déjà une signification.

Il est possible de faire la même remarque que pour l'opération « dézoomer » ; si l'utilisateur veut déplacer la vue vers le bas, il faut d'abord qu'il la déplace légèrement vers le haut. Nos observations informelles indiquent que ceci semble poser moins de problèmes que la nécessité de zoomer un peu afin de pouvoir dézoomer. Ceci est probablement dû au fait que le changement d'échelle du zoom est généralement vu comme une opération relative (typiquement contrôlée par le déplacement d'un curseur sur un potentiomètre autour d'une origine) tandis que le positionnement sur une surface plane est une opération d'une autre nature (et qui s'applique directement sur la surface affichée via la notion de pointage).

Un Control Menu est bien adapté pour contrôler deux paramètres intégraux. Deux paramètres sont intégraux (Jacob and Sibert, 1992) lorsque leurs attributs se combinent dans la pensée de l'utilisateur en un seul attribut composé. Par exemple, les coordonnées x et y d'un objet sont intégrales car les utilisateurs les combinent et les considèrent comme la position de l'objet. Un déplacement diagonal de la souris a alors une signification simple : déplacer l'objet sur la diagonale. Par contre, la taille et la couleur d'un objet ne sont pas deux paramètres intégraux. Si un Control Menu était utilisé pour contrôler de tels paramètres simultanément, un déplacement diagonal de la souris n'aurait pas de signification immédiate.

Boutons et sous-menus

Les Control Menus peuvent également contenir des commandes simples qui n'ont pas de paramètre. Dans ce cas les commandes qui peuvent être annulées

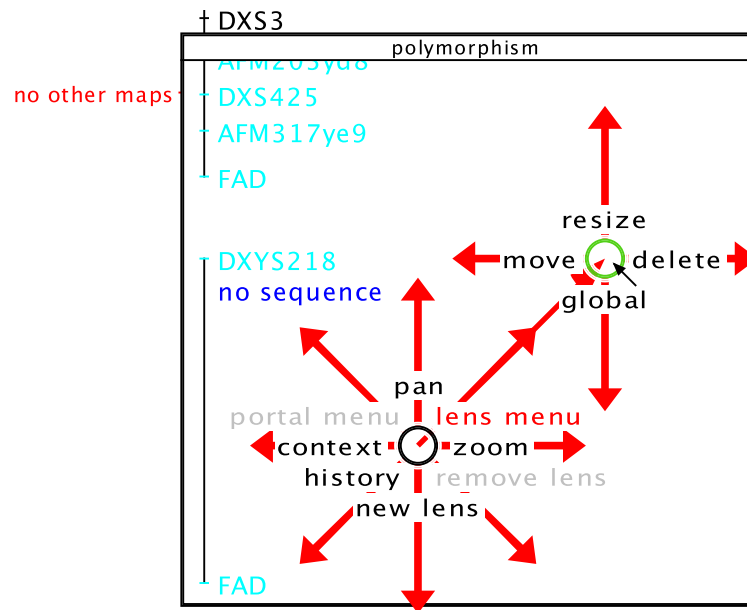


FIG. 9.8 – Sous-menu des lentilles dans notre interface zoomable

sont exécutées dès que le curseur a été déplacé de la distance d'activation depuis l'endroit où le bouton de la souris a été enfoncé. Puisque l'utilisateur maintient le bouton de la souris enfoncé, un mouvement du curseur dans l'autre direction annule l'opération. L'utilisateur peut alors infirmer l'annulation en déplaçant à nouveau le curseur dans l'autre sens. L'opération ne devient définitive que lorsque le bouton de la souris est relâché (comme avec les « boutons poussoirs » habituels).

Un Control Menu peut avoir des sous-menus. L'utilisateur enfonce le bouton de la souris et déplace le curseur dans la direction de l'entrée dans le Control Menu correspondant au sous-menu désiré. La Figure 9.8 montre le sous-menu qui permet à l'utilisateur de contrôler des lentilles affichées dans la ZUI. Ce sous-menu peut être affiché lorsque le pointeur est sur une lentille (la lentille « polymorphisme » dans la Figure 9.8). Le sous-menu est visuellement attaché au menu principal par une ligne rouge avec deux flèches et l'utilisateur choisit une opération dans le sous-menu en déplaçant le pointeur dans la direction de son entrée dans le sous-menu.

Control Menus versus marking menus

Avec un marking menu la distance parcourue par le curseur n'a pas d'importance. Seule la forme du mouvement est significative, et celle-ci est analysée une fois que le bouton de la souris relâché (ou quand l'utilisateur arrête de bouger le

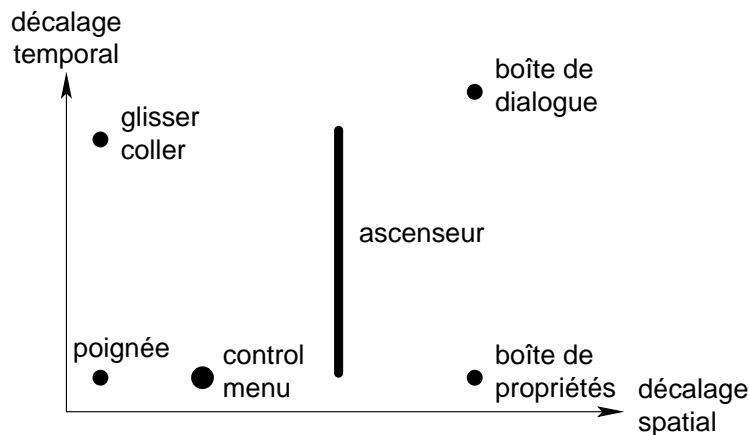


FIG. 9.9 – Degré d'indirection

curseur afin d'ouvrir un sous-menu). Avec un Control Menu la distance parcourue par le curseur est informative. La position du curseur est constamment analysée et l'opération commence dès que la distance d'activation a été atteinte.

Analyse des propriétés des Control Menus

Le modèle d'interaction de Beaudouin-Lafon (2000) définit un espace d'analyse qui peut être utilisé pour comparer de nouveaux interacteurs aux interacteurs déjà connus.

Un Control Menu est contextuel : il agit soit sur l'objet, soit sur la position se trouvant sous le curseur. Il ne nécessite pas non plus de se déplacer vers une barre de menu. L'*écart spatial* (la distance sur l'écran entre le menu et l'objet sur lequel il agit) est donc nul. Un Control Menu ressemble aux poignées des objets dans un logiciel de dessin graphique ou à l'action de glisser/coller (« drag and drop »). Lors du zoom et du défilement, l'écart temporel (le temps entre le mouvement de la souris et la réaction de l'objet manipulé) est également nul car l'interface réagit immédiatement quand l'utilisateur déplace la souris. De ce point de vue, un Control Menu ressemble à une poignée ou à une barre de défilement (si elle réagit immédiatement aux mouvements de la souris). Le degré d'indirection (qui combine ces deux écarts) est donc faible. La Figure 9.9 montre comment le degré d'indirection d'un Control Menu peut être comparé avec ceux d'autres interacteurs.

Le rapport entre le nombre de degrés de liberté fourni par notre interacteur et le nombre de degrés de liberté de la souris (le degré d'intégration) est de 2/2 car, pour toutes les interactions dans notre ZUI, les deux degrés de liberté de la souris sont utilisés. Pour faire défiler la vue dans un logiciel avec des barres de

défilement, il est nécessaire d'en utiliser deux car chaque barre n'utilise qu'un des deux degrés de liberté de la souris. Par contre, un Control Menu utilise les deux degrés de liberté de la souris. Un seul Control Menu est donc suffisant pour faire défiler la vue dans toutes les directions.

Le degré de compatibilité (la similarité entre l'action physique de l'utilisateur et la réponse de l'objet sur lequel il agit) est élevé pendant un défilement car les mouvements de la souris sont directement reflétés par les mouvements de la vue. L'utilisateur déplace l'objet sous le curseur au début de l'opération jusqu'à sa nouvelle position. Cette action présente un même niveau de compatibilité que le glisser/coller ou que de tirer sur une poignée. Ce degré de compatibilité est moins élevé pendant un zoom du fait de la possibilité d'annulation. En effet, tandis que les mouvements horizontaux sont en relation directe avec les mouvements de la souris, l'annulation requiert des mouvements verticaux, ce qui peut paraître moins évident à l'utilisateur. Des utilisateurs ont également suggéré que le choix des mouvements horizontaux pour contrôler le niveau de zoom était sous-optimal et que des mouvements verticaux seraient plus naturels pour réaliser cette opération.

Cette analyse permet de constater que les Control Menus partagent de nombreux attributs avec les poignées des objets graphiques.

Enfin, un Control Menu a les mêmes avantages qu'un pie menu en ce qui concerne la loi de Fitts. L'utilisateur n'a pas besoin de bouger la souris pour faire afficher le menu, le menu s'affiche sous le pointeur, et les zones de l'écran sur lesquelles il faut déplacer le pointeur pour sélectionner une opération sont grandes et proches du pointeur.

Applications

Un Control Menu est un nouveau type de menu contextuel qui peut être utile pour de nombreuses applications. Nous l'avons utilisé dans deux exemples de ZUIs et dans une interface de monde virtuel. Nous donnons aussi des indications sur la manière dont il pourrait être utilisé dans un traitement de texte.

Interaction dans un monde virtuel Un Control Menu a été utilisé pour se déplacer dans le monde virtuel *vrng* (Figure 9.10) disponible à l'URL <http://www.infres.enst.fr/net/vrng/>. Ce Control Menu a une forme légèrement différente de celle utilisée dans les ZUIs car ici le Control Menu est principalement utilisé pour se déplacer. Il n'est donc pas nécessaire d'afficher du texte dans le menu, les opérations disponibles (qui contrôlent la vitesse de déplacement dans diverses directions) étant indiquées par des flèches (Figure 9.10b). À la différence du cas précédent, un guide visuel est affiché une fois l'opération sélectionnée. Ce guide indique la liste des vitesses disponibles et, par conséquent, la distance nécessaire de déplacement de la souris pour choisir la vitesse souhaitée.

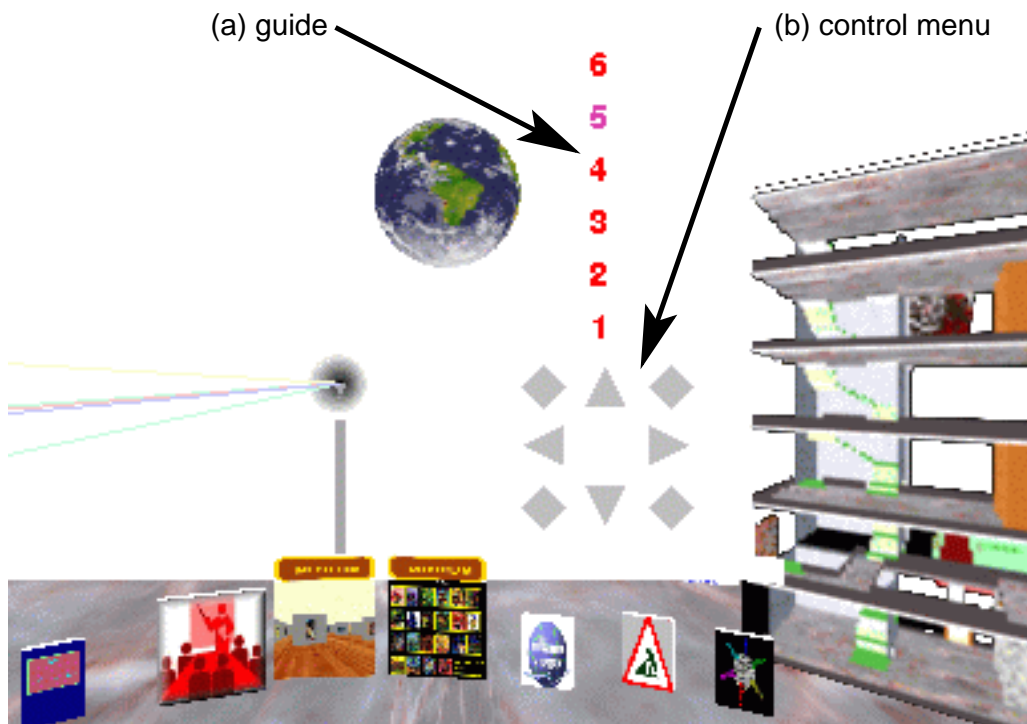


FIG. 9.10 – Le Control Menu dans vrend ; (a) le guide, et (b) le Control Menu

Traitements de texte et logiciels de présentations Dans ces logiciels, la souris est typiquement utilisée pour sélectionner du texte. Un Control Menu permet de choisir une opération parmi plusieurs opérations puis de contrôler celle-ci. On pourrait ainsi sélectionner et modifier du texte en un seul geste.

Quand un Control Menu est utilisé pour un logiciel de ce type, l'opération « sélectionner » est placée à la droite du Control Menu (c'est-à-dire qu'elle remplace l'opération « zoom » de la Figure 9.4). Dans la Figure 9.11a l'utilisateur a indiqué le début du texte devant être sélectionné en enfonceant le bouton de la souris lorsque le pointeur était entre les mots « might » et « bear ». Il déplace alors le pointeur de la distance d'activation vers la droite. À ce moment l'opération de sélection de texte commence et le texte situé entre la position initiale du pointeur et sa position actuelle est sélectionné (Figure 9.11b). La sélection de texte se fait alors normalement jusqu'à ce que l'utilisateur relâche le bouton de la souris.

Les opérations nécessitant de fournir des paramètres (des tailles de police par exemple) sont décomposées en deux gestes. Le texte à modifier est sélectionné comme décrit ci-dessus puis le menu est à nouveau utilisé pour exécuter l'opération de changement de taille.

Des opération telle que « couper », « souligner », et « mettre en gras » peuvent

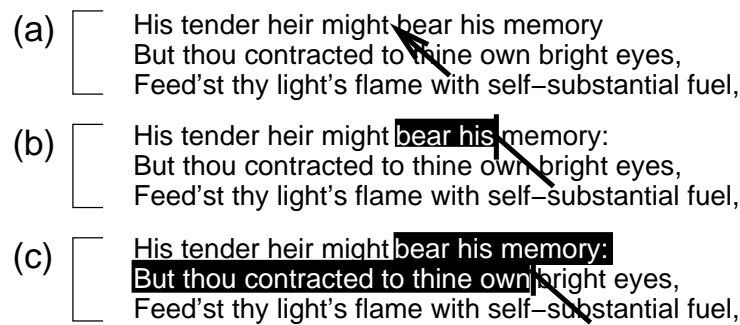
- 
- (a) [His tender heir might bear his memory
But thou contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
- (b) [His tender heir might **bear his** memory:
But thou contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
- (c) [His tender heir might **bear his memory:**
But thou contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,

FIG. 9.11 – Sélectionner du texte avec un Control Menu

être réalisées de la même manière (deux utilisations du menu) ou via une seule entrée dans le Control Menu. Dans ce dernier cas, l'opération choisie sera effectuée au fur à mesure (cas de la mise en gras) ou à la fin (cas du « couper ») de la sélection de texte. Ceci donne un retour de contrôle encore plus direct et permet d'exécuter des commandes simples en minimisant l'usage des menus.

Un Control Menu pourrait également être utilisé dans un logiciel de préparation de présentations. Ce cas est un peu différent du cas précédent dans la mesure où la taille et forme du texte sont choisies plus librement dans une présentation afin que l'utilisateur puisse créer l'effet visuel souhaité. Dans le cas d'un traitement de texte, la taille et la forme sont souvent précisées par un style pré-défini. Un guide visuel serait alors nécessaire pour choisir le style approprié, comme dans le cas du monde virtuel. Par contre, dans le cas d'un éditeur de présentation, l'utilisateur pourrait directement changer la taille des polices de manière relative comme dans le cas des interfaces zoomables. Le retour visuel est alors immédiat, ce qui permet de choisir l'effet voulu en une seule interaction. Les autres paramètres de formatage (couleur du texte, mise en gras, indentation, etc.) pourraient également être sélectionnés de la même façon (via l'utilisation ou non d'un guide visuel selon le cas).

9.2.3 FlowMenus

Les FlowMenus ont été développés après les Control Menus. Ils sont une extension des pie et marking menus et sont optimisés pour les systèmes interactifs contrôlés par un stylo. La différence principale est que le choix d'une opération est indiqué par le retour du curseur au centre de menu. Ceci permet d'exécuter plusieurs d'opérations séquentiellement sans soulever le stylo. Une autre possibilité est le contrôle de l'opération choisie. Ce contrôle peut être accompli avec un sous-menu où l'utilisateur peut sélectionner des paramètres ou par manipulation directe. La Figure 9.12 montre les mouvements (normalement invisibles) du

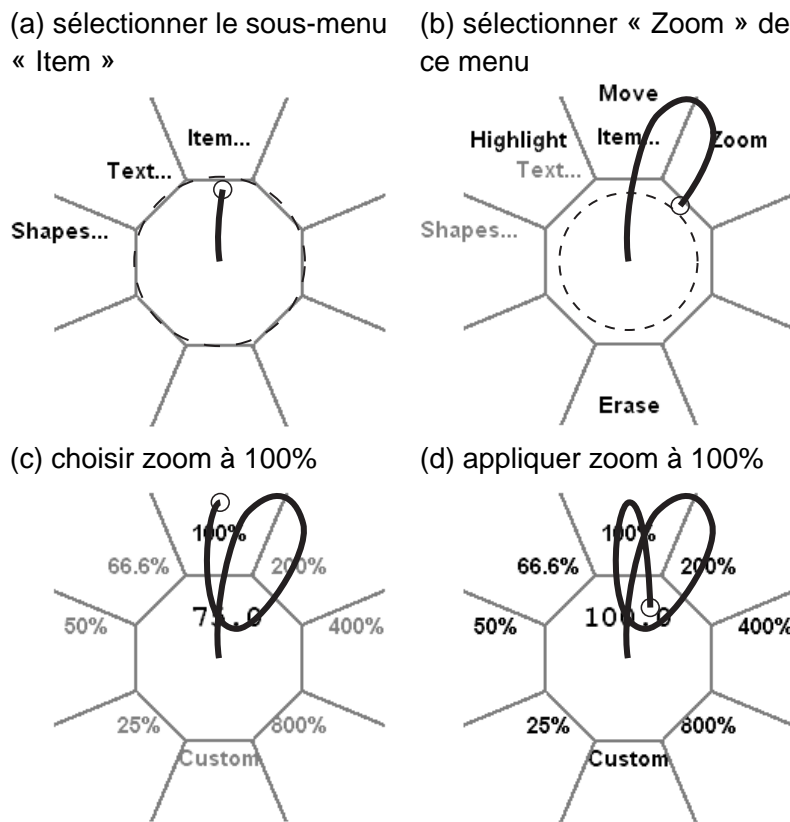


FIG. 9.12 – Échelles fixes dans un FlowMenu (copié de Guimbretière and Winograd, 2000)

stylo pendant l'utilisation d'un FlowMenu pour choisir l'opération de zoom d'un sous-menu et le choix d'un niveau de zoom de 100%. À la fin de cette opération, le stylo est toujours sur la surface et l'utilisateur peut donc choisir un autre niveau de zoom en bougeant le stylo vers la région d'un autre échelle puis de nouveau vers le centre du menu. Cette façon d'utiliser un FlowMenu ne permet que la sélection parmi une liste pré-définie. Une autre façon d'utiliser un FlowMenu ressemble à l'utilisation d'une poignée que l'on peut tourner. Ceci permet la sélection par pas prédéfini parmi un grand nombre de valeurs. Dans la Figure 9.13, l'utilisateur a décidé de choisir une échelle sur mesure. À partir de ce moment, jusqu'à ce que le stylo soit enlevé de la surface, des mouvements dans le sens des aiguilles d'une montre autour du centre du menu augmentent l'échelle chaque fois que le curseur traverse un segment. Des mouvements dans l'autre sens diminuent l'échelle.

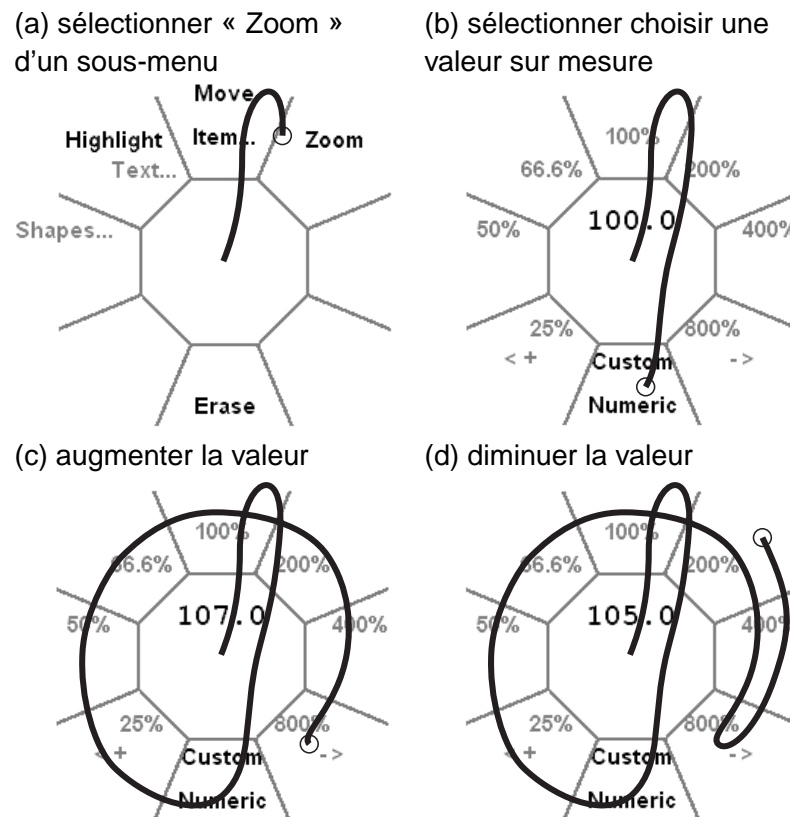


FIG. 9.13 – Valeur variable dans un FlowMenu (copié de Guimbretière and Winograd, 2000)

9.3 Visualisation

9.3.1 Recherche en visualisation

Les utilisateurs doivent comprendre et contrôler des bases de données sans cesse plus grandes et plus complexes de par la variété des types de données qu'elles contiennent. De nombreux systèmes de visualisation ont été développés pour aider les utilisateurs dans ces tâches. Cependant, les espaces d'information sont maintenant si grands que les techniques traditionnelles de présentation d'information ne sont plus suffisantes pour représenter globalement ces espaces sur un écran d'ordinateur.

La partie de l'espace d'information qui peut tenir sur l'écran est une fraction tellement réduite de l'espace global que l'utilisateur ne peut pas facilement maintenir une compréhension de sa perspective dans cet espace. Pour être utilisable un système de visualisation doit donc fournir suffisamment de contexte pour permettre à l'utilisateur de se repérer.

Selon Bartram et al. (1995), trois catégories de techniques sont utilisées pour résoudre ce type de problème : l'association du défilement et du zoom, les vues multiples, et les vues déformées.

Un système de défilement et zoom (traditionnel) impose une échelle unique, mais pas nécessairement constante, pour la vue de l'utilisateur. Le problème classique de ce type de technique est que l'utilisateur sera vite perdu en présence d'un espace d'information de grande taille car il n'a pas d'information contextuelle pour l'aider à comprendre comment sa vue se positionne dans l'espace global.

Les systèmes de vues multiples fournissent une fenêtre globale plus une ou plusieurs vues détaillées. Les vues détaillées peuvent montrer plusieurs régions de l'espace d'information ou la même région mais avec des représentations différentes. Cette solution demande de l'espace écran supplémentaire (pour des fenêtres additionnelles) et nécessite que l'utilisateur intègre mentalement les différentes fenêtres en une compréhension globale de leur position dans l'espace d'information.

Les vues déformées (ou vues « fisheye » ou œil de poisson) distordent l'espace d'information afin de montrer de manière détaillée quelques régions spécifiques (le focus) tout en gardant visible le contexte dans la même fenêtre. Nous présentons ci-après quelques techniques qui utilisent des vues déformées et expliquons pourquoi elles ne constituent pas toujours la solution idéale.

La visualisation des arbres ou des hiérarchies est le sujet de nombreuses études. Nous présenterons une description des systèmes de visualisation pour les arbres. Celle-ci servira comme exemple de la visualisation d'un type de données précis. Ces systèmes seront utilisés pour illustrer les taxonomies présentées à la fin de ce chapitre.

D'autres systèmes utilisent des vues transparentes. L'utilisation de ce type de technique introduit une nouvelle dimension dans l'interface homme machine : la profondeur. L'utilisation de la profondeur permet à plusieurs vues d'être visibles au même moment sur le même espace écran. Ces systèmes nécessitent des techniques spécifiques pour faciliter la séparation des différentes vues.

Les interfaces à échelle variable, nommées également interfaces zoomables ou « Zoomable User Interfaces » (ZUIs), sont dérivées des interfaces traditionnelles de défilement et zoom. Ces interfaces permettent aux utilisateurs de contrôler directement l'échelle de la représentation de l'espace d'information. Les ZUIs sont basées sur le principe de zoom sémantique : la représentation des objets change afin de montrer ceux-ci en plus de détail en fonction de l'espace qui leur est alloué sur l'écran. Cette technique se distingue des techniques de vues multiples car elle est immersive. Lorsqu'un utilisateur change l'échelle, la vue entière change d'échelle et l'utilisateur « plonge » dans l'espace d'information.

Les techniques traditionnelles de défilement/zoom et de fenêtres multiples sont bien connues et sont largement utilisées dans les systèmes commerciaux ac-

tuels. Elles ont cependant montré leurs limites. Nous avons donc choisi de nous concentrer sur des techniques nouvelles, principalement les ZUIs, et sur les façons de les utiliser pour aider à résoudre les problèmes de visualisation. Nous présentons ci-après les techniques mentionnées ainsi qu'une analyse de ces techniques par rapport à quelques taxonomies existantes. Nous introduirons enfin deux nouvelles taxonomies. La première analyse comment un système interactif peut présenter de l'information : par un multiplexage du temps, de l'espace, ou de la profondeur. La seconde décrit les différents types de déformations qui peuvent être utilisés pour faire du multiplexage.

Vues déformées

Les techniques présentées dans cette section sont fondées sur l'idée que les utilisateurs ont un focus d'attention (c'est-à-dire l'objet ou les objets sur lesquels ils sont en train de travailler actuellement), pour une tâche donnée ou à un moment donné, et que les autres éléments (ceux hors du focus) sont moins intéressants. Ces objets non-focaux sont cependant importants car ils permettent aux utilisateurs de positionner le focus dans l'espace d'information global. Une représentation de ces objets est nécessaire comme contexte du focus. Elles supposent également que l'intérêt de l'objet est inversement proportionnel à la distance entre le focus et l'objet. Ces techniques créent donc une correspondance entre l'espace accordé aux objets et leur distance par rapport au focus. Plus une information est éloignée du focus, moins elle est supposée être intéressante et moins il lui est alloué d'espace. Ces techniques déforment la représentation de l'espace en éliminant de l'information ou en changeant la taille ou la position de la représentation de l'information. La déformation n'est pas constante sur l'espace d'information : les informations loin du focus sont plus déformées que celles qui en sont proches. Il est possible de généraliser ces techniques pour prendre en compte la possibilité d'avoir plusieurs focus d'attention. Ceci nécessite des fonctions de distance et d'estimation de focus d'attention plus complexes.

Vues fisheye Les vues fisheye (Furnas, 1986) constituent une façon d'intégrer le contexte et le focus dans une vue unique.

Une fonction de niveau d'intérêt (DOI) donne à chaque point dans la structure devant être visualisée une valeur qui indique le niveau d'intérêt de l'utilisateur pour ce point pour une tâche donnée. Une stratégie simple d'affichage d'information, sur un écran de taille n , affiche les n points les plus intéressants selon la fonction DOI. La généralisation de ces vues par Furnas (1986) décompose la fonction DOI en deux composants : $d_f(x \vdash y) = i_a(x) - D(x, y)$, où $i_a(x)$ est l'intérêt *a priori* dans un point x , $D(x, y)$ est la distance entre x et y , et $d_f(x \vdash y)$ le niveau d'intérêt de x si le focus actuel est y . L'intérêt décroît avec la distance : si

```

1  #define DIG 40
... 2  #include <stdio.h>
4  main()
5  {
... 6      int c, i, x[DIG/4], t[DIG/4], k = DIG/4, noprint = 0;
8      while((c=getchar()) != EOF){
... 9          if (c >= '0' && c <= '9'){
16         } else {
17             switch(c) {
... 18                 case '+':
27                 case '-':
... 38                 case 'e':
⇒ 39                     for(i=0;i<k;i++) t[i] = x[i];
40                     break;
... 41                 case 'q':
... 43                 default:
46             }
... 47             if(!noprint){
57             }
58         }
59         noprint = 0;
60     }
61 }

```

FIG. 9.14 – Vue fisheye d'un programme C (copié de Furnas, 1986)

la distance au focus est grande, l'information ne sera affichée que si elle est très intéressante. Cette formulation peut être utilisée avec tout type de structure où les fonctions $i_a(x)$ et $d(x,y)$ peuvent être définies.

La Figure 9.14 montre une vue Fisheye d'un programme C (Furnas, 1986). Les numéros de ligne dans le programme sont affichés sur la gauche de la figure et la ligne courante est indiquée par « \Rightarrow ». Les « ... » indiquent où des lignes ont été enlevées. Tout le code dans le `case` courant est affiché, ainsi que les structures de contrôle qui enferment la ligne courante et les déclarations de variables accessibles à cette ligne. D'autres structures de contrôle (les lignes 47 et 57) dans le bloc de code source à côté de la ligne courante sont également affichées.

Table Lens Les Table Lenses (Rao and Card, 1994, 1995) sont un moyen de visualiser et comprendre les grands tableaux, du type de ceux que l'on peut trouver dans les tableaux. Cette technique peut être vue comme une adaptation du principe des « feuilles en caoutchouc » (sous-section 4.1.4) aux tableaux. Elle aligne et

Year	Product		Quarter	Channel	Units	Revenue	Profits
1993	ForeCode Pro						
1992	ForeWord Pro	539	1	VAR	1	226	79
		540	1	Retail	16	3200	961
		541	1	Retail	12	2400	720
		542	1	Retail	5	1000	300
	ForeMost Server						
	ForeMost Lite						
	ForeMost Access	756	4	VAR	761	684900	287658
		757	4	VAR	475	427500	179550
		758	4	VAR	428	385200	161784

FIG. 9.15 – Table Lens (de www.parc.xerox.com)

retaille la région du focus au bord des cellules dans le tableur (Figure 9.15). La région du focus peut être manipulée avec trois opérateurs standard : *zoomer* pour modifier l'espace alloué à la région du focus sans modifier le nombre de cellules dans le focus, *ajuster* pour modifier le nombre de cellules dans la région du focus, et *glisser* pour déplacer la région du focus afin de modifier les cellules dans cette région.

Les Table Lenses utilisent différents types de représentations graphiques pour afficher le contenu des cellules. Le choix de la représentation dépend de la région de la cellule (dans la région du focus, dans la même colonne, dans la même ligne, hors de la région du focus) et sa taille. Par exemple, une cellule qui contient une valeur numérique affichera des chiffres si on lui a alloué suffisamment d'espace ; sinon elle sera représentée sous une forme graphique plus compacte. Ce graphique pourrait n'être qu'un pixel en hauteur, ceci donnant une idée de la valeur dans la cellule comparativement aux cellules voisines. Cette technique est une forme de zoom sémantique dans la mesure où chaque cellule ajuste sa représentation à l'espace alloué.

Interfaces zoomables

Les interfaces zoomables (« zoomable user interfaces » ou ZUIs) ne sont plus une nouveauté et leurs principes (Furnas and Bederson, 1995) et applications pra-

tiques (Bederson et al., 1996) ont déjà été présentés dans plusieurs publications. Quand un utilisateur interagit avec une ZUI, il voit une vue d'un espace d'information. La vue initiale montre l'espace à une échelle qui permet de l'afficher en entier sur l'écran de l'utilisateur. Celui-ci peut alors « zoomer » (agrandir) la partie de la vue qu'il trouve intéressante. Les objets graphiques s'agrandissent jusqu'à ce qu'il y ait suffisamment de place sur l'écran pour remplacer ces objets graphiques par des représentations alternatives montrant les données sous-jacentes avec plus de détails. Nous avons employé cette technique, nommée « zoom sémantique », pour visualiser et parcourir la base de données HuGeMap qui regroupe les principales cartes génétiques et physiques du génome humain. Cette ZUI a été utilisée pour expérimenter les nouvelles techniques décrites dans cette thèse. Pour comprendre les exemples présentés, il suffit de savoir que la première vue montre 24 chromosomes (Figure 9.16a), que ces chromosomes possèdent trois cartes génétiques (Figure 9.16b), et que ces cartes consistent en des marqueurs génétiques positionnés le long d'un axe (Figures 9.16d et 9.16e). Enfin, la séquence de chaque marqueur génétique lui est associée sous forme d'une chaîne de caractères.

Deux nouvelles taxonomies

Les deux questions importantes qu'une taxonomie doit prendre en compte lors de la classification des systèmes de visualisation sont : sous quelle forme l'information est présentée aux utilisateurs et comment les utilisateurs contrôlent cette représentation. Nous proposons deux nouvelles taxonomies qui permettent de répondre directement à ces questions.

Taxonomie de présentation Quand un système de présentation de l'information affiche simultanément plusieurs parties ou représentations de l'espace d'information, le système utilise du multiplexage de l'espace, du temps ou de la profondeur. Si l'écran est trop petit pour afficher ce qu'un système de visualisation doit afficher à un moment donné seuls le multiplexage temporel ou en profondeur restent disponibles.

Multiplexage du temps Le multiplexage du temps implique que le système présente différentes parties ou représentations de l'espace d'information séquentiellement. Ce que le système affiche à un moment donné peut être contrôlé par le système ou directement par l'utilisateur. Le désavantage principal du multiplexage temporel est que l'utilisateur doit se souvenir de ce qu'il a vu précédemment pour pouvoir le rapprocher de ce qui sera affiché ensuite.

Les outils basés sur la déformation se servent également du multiplexage temporel car certaines parties deviendraient si petites qu'elles seraient alors invisibles. L'utilisateur doit alors déplacer le focus vers ces régions pour pouvoir les lire.

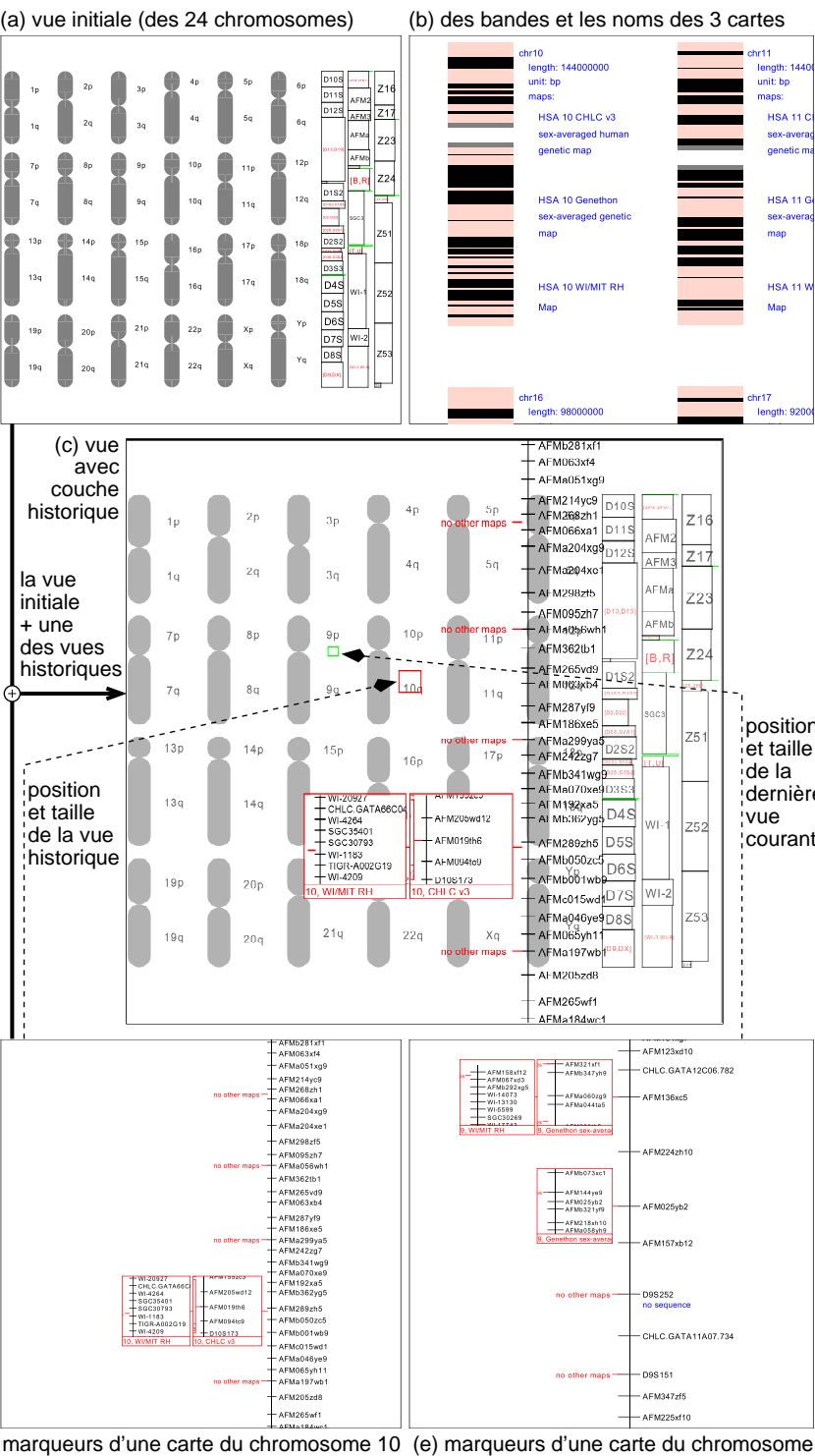


FIG. 9.16 – Vues de ZoomMap et de sa couche historique

Multiplexage en profondeur Le multiplexage en profondeur signifie que différentes parties de l'espace d'information sont superposées sur le même espace écran. L'utilisateur doit se servir des indices de profondeur pour déterminer l'appartenance des éléments graphiques aux différentes couches. L'utilisation du multiplexage en profondeur peut amener à une surcharge de l'écran, ceci demandant un effort important de la part de l'utilisateur pour distinguer les différentes couches.

Multiplexage de l'espace Le multiplexage spatial affiche deux vues différentes au même moment mais dans des espaces écran séparés. Les deux désavantages principaux du multiplexage spatial sont la quantité d'espace sur l'écran utilisé et le temps que l'utilisateur doit consacrer à déplacer sa vue d'une partie de l'écran à l'autre (et à la compréhension synthétique de l'ensemble).

Taxonomie de déformation Les systèmes de visualisation par déformation présentés dans cet état de l'art essaient de montrer l'espace d'information en entier à tout moment. L'espace sur l'écran étant presque toujours plus petit que l'espace d'information, ces systèmes utilisent donc des techniques de déformation afin de réduire la taille de la représentation. Cette déformation utilise une fonction qui représente le niveau d'intérêt de chaque donnée par rapport à la tâche de l'utilisateur et le type d'information. L'espace accordé à une information est proportionnel au niveau d'intérêt de cette information. Cette section présente les trois techniques de déformation les plus courantes.

Déformation logique Ce type de déformation nécessite une compréhension de la structure de l'espace d'information. Lorsqu'une région de l'espace d'information nécessite trop d'espace écran pour être affichée complètement, une fonction spécifique à l'application (c'est-à-dire dépendante du type d'information affichée) crée un résumé de cette région qui puisse tenir dans l'espace disponible. Ce résumé est alors affiché sans déformation physique. Ce type de déformation a été appelé « vue focus+contexte logique » par Herman et al. (2000).

Le système de visualisation des programmes C est un exemple de ce type de technique. Les parties du programme C qui ne sont pas intéressantes (c'est-à-dire distantes de la ligne courante de l'utilisateur) n'obtiennent que peu d'espace sur l'écran. Elles sont résumées par une fonction capable d'analyser des programmes C. Cette fonction convertira, par exemple, les blocs de contrôle qui entourent la ligne courante mais en sont un peu distantes, en une seule ligne comprenant uniquement le `for` ou `while` qui contrôle le bloc. Si l'espace disponible le permet, les lignes qui déclarent les variables dans le bloc seront également affichées. Les lignes appartenant au focus sont affichées comme du texte sans déformation.

Un autre exemple de déformation logique est la Table Lens. Dans ce système l'utilisateur décide quelles sont les lignes et colonnes qui sont importantes. Une taille suffisante est allouée aux lignes et colonnes pour afficher leur valeur en format numérique. Aux autres lignes et colonnes (celles qui ne font pas partie du focus et sont donc moins intéressantes pour l'utilisateur) est alloué moins d'espace et elles sont représentées de manière plus compacte.

L'avantage de ce type de déformation vient du fait que la déformation est adaptée et optimisée pour le type de données à visualiser. Cette déformation évite les déformations pseudo-optiques qui demandent souvent des ressources de calcul importantes et rendent souvent le texte ou les graphiques illisibles.

Le désavantage de ce type de déformation vient de la nécessité d'avoir une fonction de déformation qui comprenne la structure de l'espace d'information. Une nouvelle fonction de déformation doit donc être développée pour chaque type d'espace d'information.

Déformation de la position Une autre façon de déformer un espace d'information consiste à modifier la position et la taille des objets dans l'espace. Ce type de déformation laisse les objets intéressants (ceux du focus) au centre de la fenêtre tout en poussant les objets hors focus vers les bords. Les objets sont positionnés de telle sorte les objets du focus soient suffisamment écartés pour être étiquetés avec l'information demandée. Les étiquettes des objets ne sont pas déformées et restent donc lisibles. Ce type de déformation est appelé « géométrique » par Herman et al. (2000).

Ce type de déformation a l'avantage d'exiger une fonction de déformation nécessitant une moindre connaissance de l'espace d'information. Dans le cas de la déformation logique, la fonction de déformation n'a besoin que de résoudre un problème de positionnement. Contrairement à une fonction de déformation logique, une fonction de déformation de la position doit simplement comprendre la structure de l'information.

Un désavantage de ce type de déformation est que les relations spatiales entre les objets sont souvent perdues. Ce problème est exacerbé par le fait que l'utilisateur n'a souvent à sa disposition que peu d'informations qui peuvent l'aider à comprendre la déformation.

Déformation pseudo-optique Une déformation pseudo-optique n'élimine pas d'information de l'affichage et ne repositionne pas les objets dans l'espace d'information. Elle se contente de déformer globalement une image de l'espace d'information qui serait trop grande pour pouvoir tenir dans la fenêtre de l'utilisateur et pour que le focus d'attention de celui-ci reste est lisible.

Ce type de déformation a l'avantage de ne pas exiger de compréhension de

logique	déformation	
	positionnel	pseudo-optique
vue fisheye	affichage hyperbolique	Perspective Wall
Table Lens	Rubber Sheet	Document Lens
zoom sémantique		zoom sémantique
		3D Pliable Surfaces
		transformation non-linéaire

TAB. 9.1 – Taxonomie de modes de déformation

type d'interface	multiplexage	mode de déformation
défiler & zoomer	temporel	optique
fenêtres multiples	spatial	aucun
basé distorsion	temporel & spatial	tous
transparence	en profondeur	positionnel
zoom sémantique	temporel	optique & logique

TAB. 9.2 – Sommaire de la taxonomie de visualisation

l'espace d'information. Il est également indépendant de la disposition de l'espace d'information. De plus, l'espace entier est déformé, ce qui facilite la compréhension de la déformation par l'utilisateur.

Le désavantage principal de la déformation pseudo-optique est que les éléments graphiques et textuels sont souvent tellement déformés qu'ils deviennent difficilement lisibles.

Résumé Les systèmes de visualisation d'information utilisant la déformation peuvent être classifiés dans une des catégories ci-dessus. Cette classification est résumée dans la Table 9.1. Les déformations logiques sont plus difficile à mettre en œuvre mais comportent l'avantage de créer des vues où le texte et les graphiques restent lisibles.

Conclusion

Les techniques de visualisation présentées peuvent être divisées en cinq types. Ceux-ci utilisent une des trois stratégies pré-citées afin d'afficher un espace d'information dans une fenêtre de taille réduite. Un sommaire de ces techniques et des stratégies associées se trouve dans la Table 9.2. Les techniques basées sur du

zoom sémantique (les interfaces zoomables) utilisent un multiplexage du temps. La section suivante explique comment combiner le multiplexage spatial, temporel et en profondeur pour créer des interfaces zoomables plus puissantes.

9.3.2 Nouvelles aides de contexte pour les interfaces zoomables

Avec une ZUI, un utilisateur ne voit qu'une vue à la fois : le focus. Le contexte a été perdu. D'autres types d'interfaces telles que les vues « fisheye » (Furnas, 1986) et le « Document Lens » (Robertson and Mackinlay, 1993) intègrent le focus et le contexte en affichant une partie de l'information qui entoure le focus. Ces techniques déforment la représentation graphique de l'espace d'information en éliminant certains objets ou en modifiant leur taille ou leur position. D'autres techniques proposent une vue du contexte affichée à côté de la vue du focus (un multiplexage spatial) ou bien une vue du contexte affichée à la place de la vue du focus (un multiplexage temporel).

Couche de contexte

Contrairement à ces techniques, nous proposons une couche de contexte qui combine le focus et son contexte dans une seule fenêtre et sans déformation. Ceci peut être vu comme un multiplexage de la profondeur (Cox et al., 1998). La couche de contexte est temporaire et affichée uniquement quand l'utilisateur le désire. Pendant son utilisation elle se superpose en transparence à la vue principale (le focus). L'affichage de cette couche est temporaire afin de ne pas surcharger l'écran quand l'utilisateur n'a pas besoin de voir le contexte. Elle disparaît dès que l'utilisateur termine le geste qui a provoqué l'apparition de cette couche à l'écran.

La couche de contexte peut être contrôlée dans deux directions : l'échelle du contexte (c'est-à-dire le niveau de zoom sémantique) et le niveau relatif de transparence des deux vues. Le réglage de l'échelle permet de montrer une vue contextuelle dont l'échelle peut varier de manière continue entre celle de la vue initiale et celle du focus courant. Ceci permet d'obtenir une « quantité de contexte » adéquate pour une vue focale donnée.

La Figure 9.16d montre une vue que l'utilisateur peut voir après avoir navigué pendant quelques temps dans la ZUI d'une base de données biogénétiques. Cette vue ne contient pas d'élément permettant à l'utilisateur de savoir sur quelle carte de quel chromosome il se trouve. L'utilisateur peut alors afficher la couche de contexte, ce qui donne la Figure 9.17a. Celle-ci est une superposition du contexte (Figure 9.16a) sur le focus (Figure 9.16d). La position du focus relative au contexte est indiquée par un rectangle situé au centre. Dans la Figure 9.17a ce rectangle est sous le mot « 10q » et l'utilisateur sait donc que le focus montre une partie du chromosome 10. Dans la Figure 9.17b, l'utilisateur a zoomé la couche de contexte

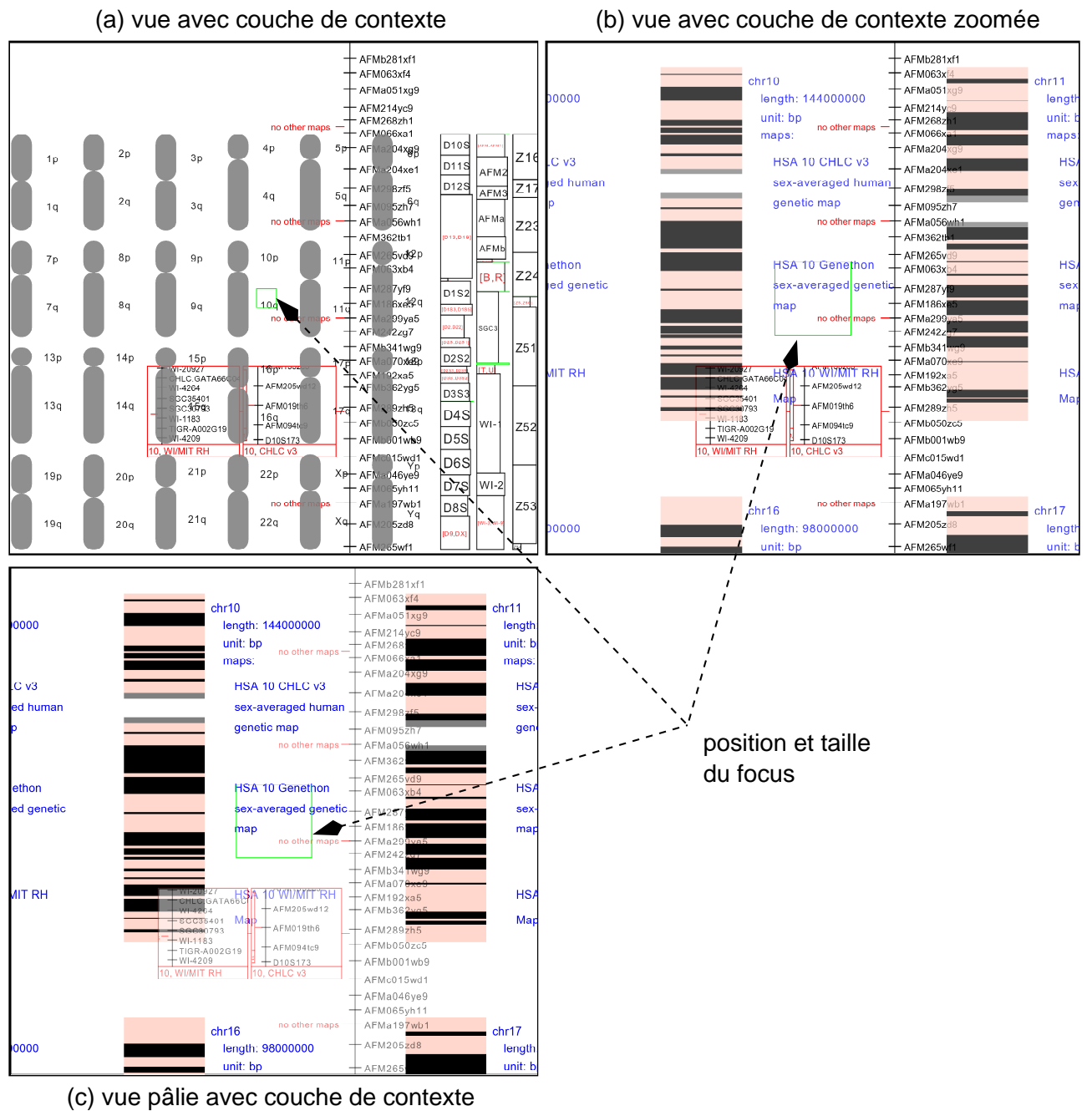


FIG. 9.17 – Construction de la couche de contexte

(mais pas le focus) ce qui donne maintenant comme contexte une vue où les noms des cartes génétiques sont visibles. L'utilisateur peut voir le rectangle (indiqué par une flèche) qui rappelle la position du focus sur le texte « Généthon ». L'utilisateur sait donc que le focus montre actuellement cette carte génétique. L'utilisateur a pu voir le contexte à deux échelles différentes de manière à pouvoir situer le focus dans deux contextes différents.

Notre système permet d'autre part à l'utilisateur de se concentrer sur l'une des deux vues en changeant la transparence relative de ces deux vues. La transparence peut varier continuellement d'un état où seul le focus est visible jusqu'à l'état opposé où seule la couche de contexte est visible. Par exemple, la Figure 9.17c est similaire à la Figure 9.17b sauf que l'utilisateur est maintenant en train de se concentrer sur le contexte et a légèrement fait disparaître le focus. Le rectangle qui rappelle la position du focus est toujours visible et l'utilisateur peut voir plus clairement que le focus montre actuellement la carte Généthon.

Contrôle des couches de contexte et d'historique Un Control Menu est utilisé pour contrôler les couches de contexte et d'historique dans notre ZUI. Ainsi, la couche de contexte a-t-elle deux paramètres : l'échelle et le niveau de transparence. L'échelle est contrôlée par les mouvements horizontaux de la souris et la transparence par les mouvements verticaux. Le contrôle de la couche historique est similaire : la position sur le chemin de l'utilisateur est contrôlée par les mouvements horizontaux et la transparence par les mouvements verticaux. Ces paramètres ne sont pas intégraux car les deux paramètres pris ensemble n'ont pas un sens simple : un mouvement diagonal de la souris n'a pas de sens intrinsèque. Nous examinons actuellement si cela crée des problèmes aux utilisateurs et si l'option qui consisterait à ne prendre en compte que la composante principale des mouvements diagonaux aiderait à résoudre cette difficulté éventuelle.

Comme expliqué précédemment, les couches de contexte et d'historique n'existent que pendant l'exécution du geste qui fait apparaître le Control Menu ; dès que l'utilisateur relâche le bouton de la souris, la couche transparente disparaît.

La différenciation entre les deux vues Harrison et al. (1995b) ont testé l'utilisation de palettes transparentes d'outils sous forme d'icônes superposées sur une fenêtre d'information. Cette surimposition crée deux couches : la couche avant contenant la palette d'icônes et celle contenant la fenêtre d'information. La couche avant, la palette, est transparente afin de ne pas (complètement) cacher les objets situés sous la palette. Harrison et al. ont fait des expériences afin de trouver le niveau de transparence optimal. Leurs expériences présentaient une icône aux sujets pendant quelques secondes, puis, une palette de douze icônes. La première icône était positionnée aléatoirement dans la palette et les sujets devaient la retrouver.

Différents niveaux de transparence de la palette ont été testés afin de trouver le niveau le mieux adapté à la lecture des icones. Cette étude a montré qu'il est possible de lire et d'utiliser de telles palettes si le niveau de transparence est adapté à la tâche de l'utilisateur. Par ailleurs, Harrison et al. (1995a) ont proposé plusieurs façons de différencier deux vues surimposées sur l'écran : les vues peuvent contenir des couleurs différentes, des polices différentes, des contenus différents (par exemple graphiques ou textuels), etc.

Notre couche de contexte est différenciée de la vue du focus par le mouvement des objets sur l'écran et par leurs niveaux de transparence respectifs. Quand l'utilisateur change l'échelle du contexte les objets graphiques de cette couche bougent sur l'écran alors que ceux de la vue du focus ne bougent pas. Ce mouvement aide l'utilisateur à situer des objets, soit dans le contexte, soit dans la vue du focus. De plus, le changement interactif des niveaux de transparence n'affecte qu'une des vues à la fois, ce qui favorise également la différenciation des deux vues.

Cox et al. (1998) ont évalué un système où les utilisateurs doivent rechercher et connecter des tubes. Ce système affiche une vue détaillée, et donc partielle, de l'espace, sur laquelle est superposée une vue transparente de tout l'espace. Les utilisateurs peuvent se servir de ces deux vues pour trouver et déplacer les tubes. La vue globale est permanente ; elle est toujours affichée, même quand l'utilisateur n'en a pas besoin (ce qui peut éventuellement gêner l'utilisateur). Cette vue a un niveau de transparence fixe. L'échelle de la vue globale est également fixe ce qui empêche l'utilisation de ce système dans les environnements où il n'existe pas de vue globale (tel que le Web) ou si la différence d'échelle entre la vue d'ensemble et la vue du détail est trop importante. Les auteurs ont testé l'utilisabilité de ce système avec différents niveaux de transparence de la vue globale. Ils ont mis en évidence que les utilisateurs trouvaient la vue globale utile pour des niveaux de transparence situés entre 50% et 70%.

Notre couche de contexte ressemble à la vue globale utilisé par ces auteurs dans la mesure où les deux systèmes superposent une vue globale à une vue locale en utilisant un effet de transparence. Il est donc raisonnable de s'attendre, dans notre cas, à des performances d'utilisabilité comparables. Notre système comporte toutefois trois améliorations importantes. D'une part, notre couche de contexte existe de manière temporaire uniquement lorsque l'utilisateur en a besoin. Son espace de représentation n'est donc encombré que de façon temporaire. D'autre part, le niveau de transparence de la couche de contexte peut être modifié interactivement par l'utilisateur : il peut donc choisir un niveau optimal en fonction de sa tâche courante. Enfin, comme expliqué précédemment, l'échelle de notre couche de contexte est également variable et le « mouvement » qui en résulte facilite la différenciation des deux couches.

Ce dernier point illustre l'importance des techniques d'interaction pour améliorer la compréhension des représentations visuelles. De même que pour le con-

trôle de l'échelle ou du niveau de transparence, il est vraisemblable que le « bouclage interactif » entre les actions de l'utilisateur et la réalisation d'effets visuels immédiats facilite la perception de certaines structures. Ceci est dû au fait que le système visuel humain est généralement plus efficace pour détecter des objets en mouvement. Cet effet devrait de plus être logiquement renforcé lorsque ce mouvement est lui-même contrôlé par l'utilisateur.

Couche historique

La couche de contexte décrite ci-dessus permet à l'utilisateur de trouver une réponse à la question « où suis-je ? ». Une autre question importante est « comment suis-je arrivé ici ? ». Les ZUIs nécessitent un mécanisme d'historique pour que l'utilisateur puisse retourner aux régions déjà visitées dans l'espace d'information afin de voir ces régions en relation avec le focus et la vue initiale. Nous proposons une autre vue temporaire appelée *couche historique*. Cette vue permet de se déplacer interactivement entre la vue initiale et la vue courante en suivant le chemin emprunté par l'utilisateur. Comme la couche de contexte, la couche historique est temporaire afin de ne pas surcharger l'écran.

De même que la couche de contexte, la couche historique est affichée en superposition de la vue courante du focus. Cette couche historique est contrôlée interactivement par l'utilisateur de telle sorte que le mouvement de la souris fasse apparaître successivement toutes les vues intermédiaires que l'on a préalablement affichées pour arriver à la vue courante. La *couche historique* (Figure 9.16c) contient une des *vues* historiques (Figure 9.16d) et se superpose à la vue initiale (Figure 9.16a) ; cette combinaison s'affiche temporairement en lieu et place de la vue courante. La vue courante (Figure 9.16e) n'est donc pas visible pendant l'utilisation de la couche historique. La position et la taille de la vue historique et de la vue courante sont indiquées sur la vue initiale par des rectangles de couleurs différentes (Figure 9.16c). L'utilisateur, en se déplaçant dans la succession des vues historiques (Figure 9.16a, b, d et e) peut donc revoir son parcours dans l'espace d'information et à tout moment mettre en relation la dernière vue courante avec une vue intermédiaire de son parcours.

Vue hiérarchique

Les techniques présentées dans la section précédente facilitent la contextuelisation du focus dans l'espace d'information. Cependant, elles n'informent pas l'utilisateur sur ce qui se trouve dans d'autres régions ni sur ce qui peut être trouvé en « zoomant » davantage. Les ZUIs sont souvent utilisées pour visualiser des données organisées hiérarchiquement mais l'utilisateur ne peut généralement pas utiliser cette organisation pour naviguer dans l'espace d'information. Par exemple,

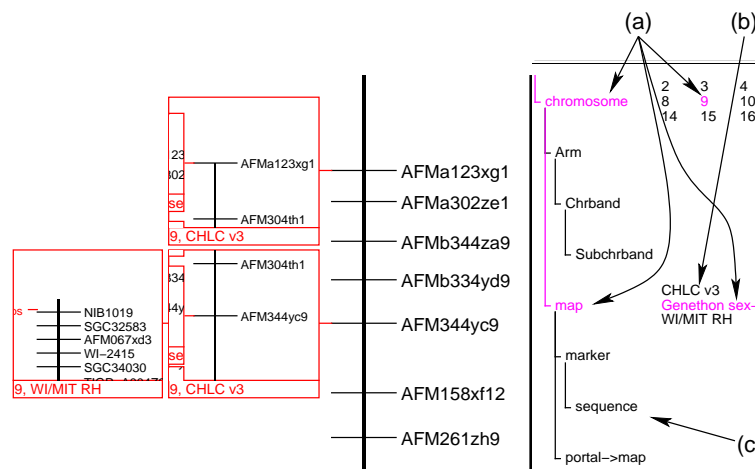


FIG. 9.18 – Notre interface zoomable avec la hiérarchie sur la droite

il est habituellement impossible de dézoomer automatiquement pour voir un objet en entier, ni d'utiliser la hiérarchie pour se déplacer d'un sous-objet vers un autre.

Une nouvelle vue hiérarchique Les ZUIs fournissent une vue qui peut être considérée comme une tranche horizontale de l'espace 3D d'information (en considérant que la dimension verticale est celle du zoom). Nous proposons une seconde vue orthogonale à la première, qui est une tranche verticale « aplatie » de l'espace d'information. Cette seconde vue (Figure 9.18) affiche les noms des objets qui sont situés au dessus de la vue courante dans la hiérarchie. Comme les objets ont également des types (un objet peut être un chromosome, une carte, une séquence, etc) la hiérarchie entière des types d'objets peut être montrée dans la seconde vue si l'espace d'information est suffisamment régulier. Sinon, seule une partie de la hiérarchie des types, centrée sur la position de l'utilisateur, peut être montrée. Cette seconde vue indique à l'utilisateur la structure de l'espace d'information, sa position courante, les autres informations disponibles, leur disposition spatiale, et comment les trouver.

Dans l'espace régulier de notre ZUI biogénétique, les chromosomes possèdent deux « bras », des données et des cartes. Cette structure est affichée dans la vue de la hiérarchie (Figure 9.18) dès que la ZUI est lancée. La position de l'utilisateur dans la structure est indiquée en gris (en magenta sur l'écran) ; dans cet exemple l'utilisateur est en train de regarder la carte « Généthon » sur le chromosome 9 (Figure 9.18a). Cette vue indique également que si l'utilisateur continue de zoomer sur la carte il trouvera les séquences des marqueurs génétiques (Figure 9.18c).

La hiérarchie offre également à l'utilisateur un moyen efficace de se déplacer des objets visibles vers les objets non encore affichés qui leur sont liés. Par

exemple, si l'utilisateur clique sur le mot « chromosome » dans la Figure 9.18, la ZUI dézoomera suffisamment pour montrer le chromosome 9 en entier. L'utilisateur pourrait également cliquer sur les mots « CHLC v3 » (Figure 9.18b), le nom de la carte à côté de la carte « Généthon » sur la chromosome 9, afin de se déplacer vers la même région sur la carte « CHLC v3 ».

Évaluation Afin d'évaluer l'efficacité de la hiérarchie proposée dans cette thèse nous avons créé une version modifiée de notre ZUI sans la hiérarchie. Huit sujets, des volontaires parmi nos collègues d'Infobiogen, ont été formés à notre ZUI. Ces volontaires étaient des informaticiens, des biologistes, et des assistants administratifs. Pendant l'évaluation il leur a été demandé de répondre à vingt-deux questions à choix multiples. La séance d'entraînement leur a montré comment répondre à ce genre de question avec et sans la vue de la hiérarchie. Les sujets ont été divisés en quatre groupes et ont traité onze questions avec aide de la hiérarchie, et onze sans. Deux groupes ont commencé par répondre à onze questions avec l'aide de la hiérarchie, les deux autres groupes ont commencé sans cette aide. Ainsi chaque question a été traitée avec l'aide de la hiérarchie par deux groupes de sujets, et sans par les deux autres groupes. De plus, dans chaque cas, les deux séries de onze questions ont été présentées dans un ordre différent à chacun des deux groupes afin d'éviter que l'ordre des questions n'influe sur les résultats.

Les questions étaient du type « quel est le nom du dernier marqueur de la carte Généthon des chromosomes 1, 8, 13 ? ». Celles-ci ne concernaient pas directement la structure de l'espace d'information mais une connaissance de cette structure (fournie par l'entraînement) aidait les sujets à répondre plus rapidement aux questions.

Pour chaque sujet nous avons calculé le temps pris pour répondre aux onze questions sans la vue hiérarchique divisé par le temps pris pour répondre aux onze questions avec la vue hiérarchique. Une valeur supérieure à 1 signifiait donc que la vue hiérarchique aidait à l'exécution des tâches demandées. La moyenne était de 1,58 avec un écart type de 0,54. Ce grand écart type venait du fait que certains utilisateurs ne connaissaient pas la structure de l'espace d'information avant cette étude. Ces utilisateurs trouvaient que la séance d'entraînement n'était pas suffisante et donc que la seconde partie des questions était plus facile. Cependant, les sujets étaient en général plus rapides avec la vue hiérarchique et les commentaires étaient positifs.

Autres techniques La technique d'« Excentric Labeling » (Fekete and Plaisant, 1999) permet d'identifier des objets sur l'écran. Cette technique étiquette, au moyen de bulles d'aide situées dans la vue principale, les objets se trouvant autour du pointeur. Nous proposons une autre technique, non intrusive, pour iden-

tifier les objets autour du pointeur. Quand l'utilisateur déplace le pointeur sur la vue principale, la vue de la hiérarchie est mise à jour en affichant le type et le nom de l'objet qui est situé sous le pointeur. Si le pointeur quitte la vue principale ou s'il n'est pas sur un objet, la vue hiérarchique indique le plus bas niveau dans la hiérarchie auquel tous les objets visibles appartiennent. Cette technique d'étiquetage présente des similarités avec les bulles d'aide dans la mesure où l'utilisateur n'a pas besoin de demander l'information explicitement. Elle a toutefois pour avantage de ne pas empiéter sur la vue principale.

L'interface du système gIBIS (Conklin and Begeman, 1988) inclut une vue globale du graphe IBIS. Cette vue montre tous les nœuds du graphe (organisés selon leur liens primaires) ainsi que l'un de leurs attributs (le sujet du nœud). Elle est généralement trop grande pour être affichée en entier ce qui nécessite d'utiliser des ascenseurs pour naviguer. Quand l'utilisateur se déplace dans le graphe, en zoomant ou en défilant, la vue globale se déplace afin de montrer la position de l'utilisateur dans le graphe. Cette vue peut également être utilisée pour se déplacer dans le graphe car l'utilisateur peut cliquer sur un nœud sur lequel il souhaite recentrer le focus. Notre hiérarchie se distingue de cette vue globale du fait qu'elle ne montre que la structure de l'espace (au lieu de l'espace en entier) et les noms des objets situés hiérarchiquement « au-dessus » des objets visibles. Ainsi, même les ZUIs de grande taille (mais structurées) peuvent tenir dans un espace réduit et des ascenseurs ne sont pas nécessaires. Comme une ZUI contient typiquement un très grand nombre d'objets, une vue exhaustive de tous les objets serait trop grande pour être utilisable.

9.3.3 L'outil de développement de ZUIs : Zomit

Les techniques d'interaction présentées dans cette thèse ont été réalisées et testées dans une ZUI nommé ZoomMap, de type client/serveur, conçue pour être utilisée sur Internet. ZoomMap visualise les données de la base HuGeMap. Celle-ci est stockée dans une base de données objet gérée par le système EyeDB (Viara et al., 1999).

ZoomMap repose sur l'outil de développement Zomit. Cet outil implémente toutes les fonctionnalités de base d'une ZUI, telles que le zoom sémantique (Furnas and Bederson, 1995), les portails et les lentilles magiques. Un portail est positionné dans le monde virtuel par le développeur d'une ZUI et contient une vue d'une autre région du monde virtuel. L'utilisateur peut zoomer et faire défiler la vue dans le portail. Une liste de lentilles magiques possibles est proposée par la ZUI et l'utilisateur peut en créer une nouvelle quand il le souhaite et ensuite la déplacer. Une lentille magique contient une transformation, propre à la lentille, de la région du monde virtuel qu'elle couvre.

Zomit consiste en un client Java et une bibliothèque C++ qui communiquent

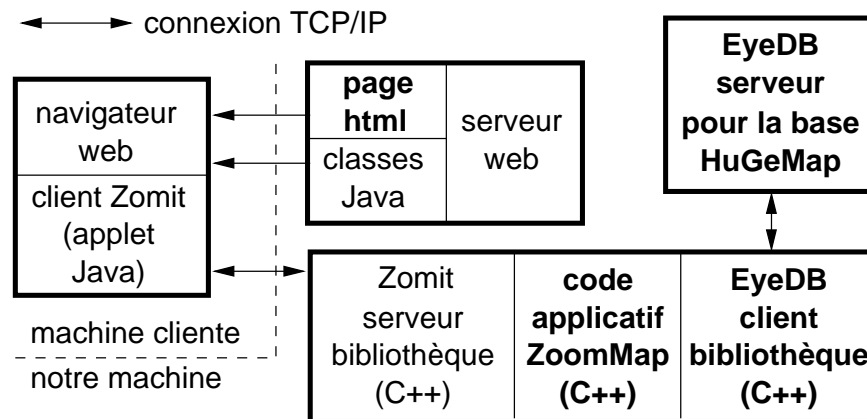


FIG. 9.19 – Implémentation client/serveur de Zomit

par une connexion TCP/IP. La Figure 9.19 montre la structure d'une ZUI construite avec l'outil de développement Zomit. Quand l'utilisateur lit une page Web qui fait référence à un client Zomit, l'*applet* Java est chargée dans le navigateur de l'utilisateur puis se connecte au serveur Zomit (qui doit se trouver sur la machine ayant envoyé la page). Le serveur Zomit lit la base de données et crée les objets du monde virtuel. Ce programme se charge de stocker ces objets et de les envoyer au client quand nécessaire.

Le client s'exécute dans un navigateur Web standard et ne nécessite pas d'installation préalable de la part de l'utilisateur. L'*applet* est complètement générique et le même code Java peut être utilisé pour communiquer avec n'importe quel serveur Zomit. L'*applet* peut aussi être installée en tant qu'application Java. Dans ce cas, le même client peut être utilisé pour communiquer avec n'importe quel serveur Zomit sur n'importe quelle machine.

Lorsque le client envoie au serveur la position de l'utilisateur dans l'espace d'information, le serveur répond en lui envoyant les objets graphiques qu'il doit afficher à cette position. Ces objets sont simples à dessiner et nécessitent peu de bande passante lors de leur transmission. Le client stocke les objets en mémoire locale et peut donc répondre rapidement aux changements d'échelle ou de position qui redemandent l'affichage de ces objets ; seul l'affichage des objets nouveaux est ralenti par la latence de la connexion avec le serveur. L'interaction avec des lentilles magiques et les portails est également gérée par le client ; seul l'affichage de nouveaux objets dans ces interacteurs peut éventuellement être ralenti.

La transparence des couches est simulée dans l'*applet* Java au moyen de fonctions XOR car les commandes graphiques utilisées pour réaliser la transparence ne sont pas encore disponibles dans de nombreux navigateurs. La fonction XOR est également nettement plus rapide que l'affichage de couches vraiment trans-

parentes par composition de couches (ou « alpha blending »). La transparence est par contre effective quand le client tourne comme une *application* Java en utilisant les bibliothèques Java 2.

Le serveur s'exécute sur la même machine que la base de données afin de pouvoir lire rapidement les nombreuses informations requises pour construire les objets graphiques à envoyer au client. De même que le client, la bibliothèque du serveur est générique ; il n'est donc pas nécessaire de la modifier pour visualiser un autre type de données. Seul le code qui lit la base de données et celui qui appelle la bibliothèque, en texte gras dans la Figure 9.19, doivent être modifiés pour une application particulière.

Le développement d'un serveur Zomit

Le serveur d'une ZUI utilise la bibliothèque C++ Zomit et doit créer, au moyen de celle-ci, des instances d'une classe. Chaque instance est enregistrée comme couvrant une région en trois dimensions (x , y , et le niveau d'échelle) du monde virtuel. Quand le client demande des objets dans une partie de la région associée à une instance (car l'utilisateur y est entré), le code de cette instance est appelé. Ce code peut créer des objets graphiques, exclusivement situés dans la région initialement associée avec l'instance, et enregistrer d'autres instances de la classe couvrant les sous-régions de la région initiale. Ces sous-régions sont normalement celles que l'utilisateur verra en zoomant davantage. Cette technique, où les objets graphiques ne sont pas générés avant d'en avoir besoin, est une forme d'évaluation paresseuse. Un serveur peut ainsi être utilisé pour visualiser un monde virtuel de très grande taille où il n'est pas possible de générer (et stocker) tous les objets graphiques par avance.

Chaque objet a une position en coordonnées absolues dans le monde virtuel et deux niveaux d'échelle entre lesquelles il est déclaré visible. L'utilisateur voit une partie rectangulaire du monde à une échelle donnée. Quand il zoome ou dézoome ce niveau d'échelle change. Ainsi quand l'utilisateur zoome sur un objet il devient de plus en plus grand jusqu'au moment où il n'est plus visible. Le concepteur peut placer les objets arbitrairement mais fera généralement en sorte qu'un objet qui disparaît soit remplacé par d'autres objets représentant la même information mais de manière plus détaillée.

Un portail est un objet semblable à un rectangle mais avec les coordonnées de la partie de monde virtuel à afficher. L'affichage du contenu d'un portail, le défilement et le zoom dans le portail sont gérés par le client sans intervention de la part du développeur.

Une lentille magique est une vue dans un monde virtuel parallèle au monde virtuel principal. Quand le développeur crée des objets graphiques, il peut préciser s'ils sont visibles dans le monde virtuel principal ou dans le monde associé à une

lentille. Les objets visibles dans les lentilles sont des objets graphiques standard, qui peuvent être stockés et affichés par le client comme des objets classiques. Ce type de lentille permet de transformer la représentation graphique des objets concernés mais ne permet pas d'y appliquer des calculs spécifiques.

9.3.4 Applications de Zomit

Zomit est un outil de développement des interfaces zoomables et a été utilisé pour créer des mondes virtuels dans deux domaines très différents. Zomit a été utilisé pour créer une interface à une base de données biogénétiques et par une autre laboratoire de recherche pour créer une interface à une bibliothèque virtuelle. Ces deux mondes virtuels peuvent être testés avec un navigateur standard aux URL <http://www.infobiogen.fr/services/zomit/> et <http://www.enst.fr/~elc/>. Zomit a également été utilisé dans des projets d'élèves pour développer des programmes d'exploration d'information.

Bibliothèque virtuelle

En plus de l'interface de la base HuGeMap, notre outil de développement d'interfaces zoomables, nommé Zomit, a été utilisé pour créer l'interface d'une bibliothèque virtuelle. Cette ZUI, qui permet de naviguer dans les rayons d'une bibliothèque, a été développée à l'École de Mines de Nantes (Lecolinet et al., 2001). L'objectif de ce projet était de comparer la vitesse de la recherche de livres en utilisant une ZUI, une interface tridimensionnelle, et une bibliothèque réelle. Cette ZUI permet de zoomer d'une vue globale de la bibliothèque (Figure 9.20) à une vue des différentes matières (Figure 9.21) puis des livres contenus dans les différents « rayons » (Figure 9.22). Cette bibliothèque contient plus de trois mille livres groupés dans des rayons, eux-mêmes regroupés par sujets. Cette interface contient également les images de couverture des livres, qui sont affichées quand l'utilisateur zoome dessus.

9.3.5 Conclusion

Nous avons présenté de nouvelles aides contextuelles, les couches de contexte et d'historique, qui sont destinées à faciliter l'orientation et la navigation des utilisateurs dans de grands espaces d'information représentés à l'aide d'interfaces zoomables. La superposition de couches transparentes permet d'intégrer le focus et le contexte dans la même vue.

Les hiérarchies présentées fournissent un nouveau moyen de combiner le focus et le contexte dans les ZUIs. Elles aident les utilisateurs à ne pas se perdre

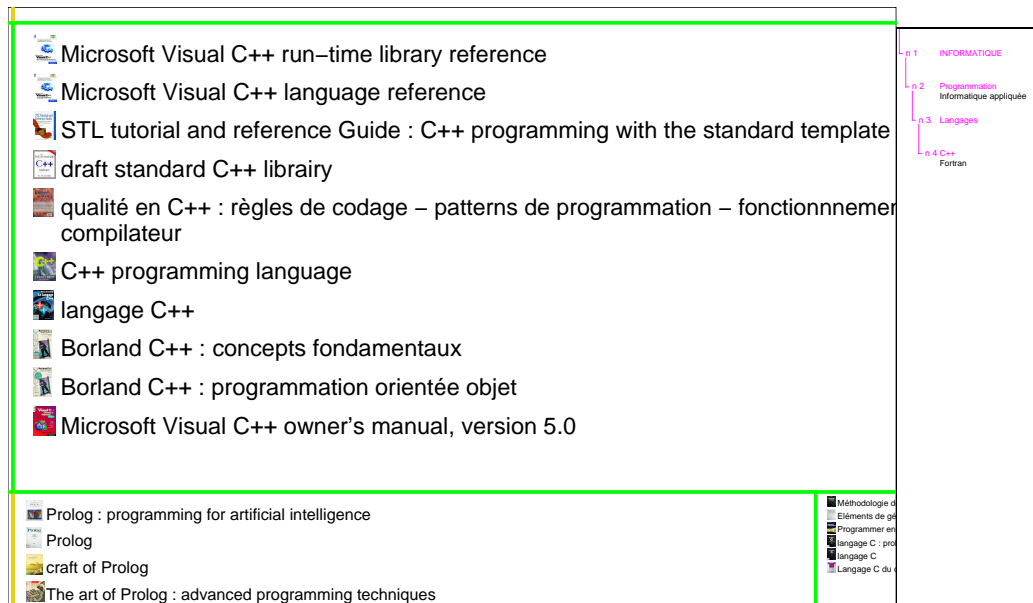


FIG. 9.22 – Une partie des livres de la bibliothèque virtuelle

dans l'espace d'information et elles les renseignent sur les informations disponibles (mais non encore visibles à un niveau de zoom donné). Une évaluation a montré l'utilité de cette technique de représentation. Des évaluations complémentaires seront nécessaires pour examiner comment les utilisateurs se servent des hiérarchies et des couches transparentes quand ces deux aides sont conjointement disponibles.

Notre nouveau Control Menu facilite la maîtrise de ces interfaces zoomables complexes.

9.4 Conclusion et perspectives

La visualisation des bases de données est importante pour de nombreuses tâches qu'elles soient liées à la recherche ou de nature commerciale. Ceci peut se résumer par la phrase : « voir aide à penser ». Les interfaces zoomables sont des outils qui permettent de créer des mondes virtuels destinés à représenter de grands espaces informationnels. Elles aident également les utilisateurs à naviguer dans ces mondes pour trouver les informations recherchées et transformer ces dernières en la représentation souhaitée. Malgré tout, la vaste taille des espaces d'information et le manque d'indices de contexte amènent souvent les utilisateurs à se perdre avant d'avoir trouvé l'information recherchée. Et, s'ils la trouvent, il est fréquent qu'ils rencontrent alors des difficultés à la situer dans l'espace d'in-

formation global.

Nous avons proposé trois nouvelles aides contextuelles qui aident les utilisateurs à comprendre l'espace d'information, leur position dans cet espace, et les liens entre les informations actuellement visibles et l'espace d'information global. La première des ces aides est une vue permanente de la hiérarchie. Cette vue montre où se trouvent les informations dans le monde virtuel et la position actuelle de l'utilisateur. La deuxième aide est une vue transparente transitoire que les utilisateurs créent lorsque cela est nécessaire et qui disparaît immédiatement quand elle n'est plus utile. Elle montre la vue actuelle en relation avec des vues plus globales. La vue globale est contrôlée interactivement par l'utilisateur de manière dynamique et continue. Cette aide permet aux utilisateurs de comprendre leur position dans le monde virtuel relativement à des vues contextuelles globales. La troisième aide est également transitoire et transparente. Elle est créée par les utilisateurs lorsqu'ils veulent connaître ou revisiter le chemin pris pour arriver à leur position actuelle dans l'espace et désirent voir ce chemin en relation avec leur vue actuelle.

Toutes les interfaces homme-machine pour les grands espaces informationnels sont confrontées au problème de l'affichage de vastes quantités d'information sur des écrans de taille réduite. De plus, afin de comprendre les relations entre différentes informations, les utilisateurs doivent pouvoir approcher des objets situés dans des endroits distants dans l'espace d'information. Il n'est pas suffisant de voir des données sous une seule représentation ; bien comprendre des données demande de les voir sous plusieurs formes à l'aide de plusieurs vues.

La première des deux solutions standard à ce problème montre plusieurs vues différentes dans des fenêtres adjacentes (multiplexage de l'espace). Cependant l'espace sur l'écran est souvent trop limité et l'intégration mentale des ces vues séparées est parfois difficile.

La seconde solution est de montrer ces vues séquentiellement ce qui force les utilisateurs à se souvenir des vues afin de pouvoir les comparer. Les lentilles magiques utilisent cette technique de multiplexage spatial car elles couvrent et cachent le focus d'attention de l'utilisateur afin de montrer cette région sous une autre représentation.

Nous avons choisi une autre technique utilisant des couches transparentes qui peuvent montrer des vues simultanément dans le même espace. Les utilisateurs peuvent utiliser des indices de profondeur afin de séparer mentalement ces deux vues. Nous avons découvert qu'un contrôle fluide et continu provoque des mouvements entre les vues qui aident les utilisateurs à mieux les séparer. Notre mécanisme de contrôle permet donc d'interagir avec deux types d'informations dans le même espace sur l'écran. Ceci réduit l'espace utilisé par un système de visualisation et facilite l'intégration des deux vues par les utilisateurs.

Nos aides contextuelles exigent une synergie entre interaction et visualisation

pour faciliter leur utilisation et augmenter leur efficacité. Cette synergie a été obtenue grâce à l'utilisation d'un nouveau type de menu, appelé « Control Menu », qui permet aux utilisateurs de choisir une opération puis de la contrôler. Cette sélection et ce contrôle sont exécutés dans un seul geste qui crée un lien entre sélection et exécution. Il évite également l'utilisation de plusieurs interacteurs pour exécuter ce qui est logiquement constitué d'une seule opération (Buxton, 1986). La fluidité d'exécution et le retour de contrôle permis par les Control Menus donnent aux utilisateurs la capacité d'obtenir et de voir le résultat souhaité avant de terminer leur geste.

Nous nous sommes concentré sur l'utilisation des Control Menus dans les interfaces zoomables en faisant également quelques essais de leur utilisation dans des environnements de réalité virtuelle. Nous pensons que ces menus peuvent être utiles dans de nombreux types d'applications, ce qui devra être validé par des évaluations auprès des utilisateurs.

Cette thèse a mis en évidence l'importance d'une synergie entre visualisation et interaction. Ce qu'un système de visualisation peut montrer aux utilisateurs est déterminé en grande partie par la manière dont les utilisateurs contrôlent le système. Un retour de contrôle immédiat aide les utilisateurs à comprendre les interactions et leur permet de faire les raffinements successifs nécessaires pour arriver au résultat souhaité.

Chapter 10

Publications

Pook, S., Lecolinet, E., Vaysseix, G., and Barillot, E. (2000a). Context and interaction in zoomable user interfaces. In *AVI 2000*, pages 227–231 & 317, Palermo, Italy. ACM Press.

Pook, S., Lecolinet, E., Vaysseix, G., and Barillot, E. (2000b). Contexte et interaction dans les interfaces zoomables. In *RJC-IHM 2000*, pages 57–60, L'Île de Berder, France. Laboratoire VALORIA (Université de Bretagne-Sud).

Pook, S., Lecolinet, E., Vaysseix, G., and Barillot, E. (2000c). Control menus: Execution and control in a single interactor. In *CHI 2000 Extended Abstracts*, pages 263–264, The Hague, The Netherlands. ACM Press.

Pook, S., Lecolinet, E., Vaysseix, G., and Barillot, E. (2000d). Des aides transparentes de navigation et un nouveau type de menu pour les interfaces zoomables. In *ERGO-IHM 2000*, pages 170–177, Biarritz, France. CRT ILS & ESTIA.

Pook, S., Lecolinet, E., Vaysseix, G., and Barillot, E. (2001). Control menus et vues contextuelles pour les interfaces zoomables. *Revue d'Interaction Homme-Machine*. In press.

Pook, S., Vaysseix, G., and Barillot, E. (1998). Zomit: biological data visualization and browsing. *Bioinformatics*, 14(9):807–814.

Chapter 11

Bibliography

Alias|Wavefront, editor (2000). *The Art of Maya*. Alias|Wavefront Education. 2.4

Arch (1992). A metamodel for the runtime architecture of an interactive system. *ACM SIGCHI Bulletin*, 24(1):32–37. The UIMS Tool Developers Workshop. 6.2.1

Arnold, K., Gosling, J., and Holmes, D. (2000). *The Java Programming Language*. Addison-Wesley, third edition. 6.3.2

Barillot, E., Guyon, F., Cussat-Blanc, C., Viara, E., and Vaysseix, G. (1998). HuGeMap: a distributed and integrated human genome map database. *Nucleic Acids Research*, 26(1):106–107. 7.1.2

Bartram, L., Ho, A., Dill, J., and Henigman, F. (1995). The continuous zoom: A constrained fisheye technique for viewing and navigating large information spaces. In *UIST '95*, pages 207–215, Pittsburgh PA, USA. ACM Press. 4, 4.18, 4.2.2, 4.2.2, 4.2.2, 9.3.1

Baudel, T. and Beaudouin-Lafon, M. (1998). Outils et méthodes de construction d'interfaces. Tutorial IHM'98. 6.2.1

Beaudouin-Lafon, M. (2000). Instrumental interaction: An interaction model for designing post-WIMP user interfaces. In *CHI 2000*, pages 447–453, The Hague, The Netherlands. ACM Press. 2.3, 2.17, 2.3, 3.5.1, 3.5.1, 9.2.2

Beaudouin-Lafon, M. and Lassen, H. M. (2000). The architecture and implementation of CPN2000, a post-WIMP graphical application. In *UIST '00*, pages 181–190, San Diego CA, USA. ACM Press. 6.3.3

- Bederson, B. and Meyer, J. (1998). Implementing a zooming user interface: Experience building Pad++. *Software: Practice and Experience*, 28(10):1101–1135. 6.4.4
- Bederson, B. B. (2000). Fisheye menus. In *UIST '00*, pages 217 – 225, San Diego CA, USA. ACM Press. 2.1.7, 2.7, 2.2.8
- Bederson, B. B. and Hollan, J. D. (1995). Advances in the Pad++ zoomable graphics widget. In *USENIX Third Annual Tcl/Tk Workshop*, Toronto, Canada. 4.4.3
- Bederson, B. B., Hollan, J. D., Perlin, K., Meyer, J., Bacon, D., and Furnas, G. (1996). Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. *Journal of Visual Languages and Computing*, 7(1):3–32. 4.4, 4.4.3, 6.4.3, 9.3.1
- Bederson, B. B., Meyer, J., and Good, L. (2000). Jazz: An extensible zoomable user interface graphics toolkit in Java. In *UIST '00*, pages 171–180, San Diego CA, USA. ACM Press. 6.4.4
- Belge, M., Lokuge, I., and Rivers, D. (1993). Back to the future: a graphical layering system inspired by transparent paper. In *CHI '93 conference companion*, pages 129–130, Amsterdam, The Netherlands. ACM Press. 4.3.1
- Bellanné-Chantelot, C. et al. (1992). Mapping the whole human genome by fingerprinting yeast artificial chromosomes. *Cell*, 70:1059–1068. 2
- Bertin, J. (1977). *La graphique et le traitement graphique de l'information*. Flammarion. 4.4.5
- Bier, E. A., Stone, M. C., Pier, K., Buxton, W., and DeRose, T. D. (1993). Tool-glass and magic lenses: The see-through interface. In *SIGGRAPH '93*, pages 73–80, Anaheim CA, USA. ACM Press. 4.3.5, 4.32, 4.33
- Buxton, W. (1986). Chunking and phrasing and the design of human-computer dialogues. In *Information Processing '86*, pages 475–480. 2.1.2, 2.4, 9.4
- Callahan, J., Hopkins, D., Weiser, M., and Shneiderman, B. (1988). An empirical comparison of pie vs. linear menus. In *CHI '88*, pages 95–100, Washington DC, USA. ACM Press. 2.1.3, 9.2.1
- Carpendale, M. S. T., Cowperthwaite, D. J., and Fracchia, F. D. (1995). 3-dimensional pliable surfaces: For the effective presentation of visual information. In *UIST '95*, pages 217–226, Pittsburgh PA, USA. ACM Press. 4.6, 4.7, 4.1.5, 4.8

- Carpendale, M. S. T., Cowperthwaite, D. J., and Fracchia, F. D. (1997a). Extending distortion viewing from 2D to 3D. *IEEE Computer Graphics and Applications*, 17(4):42–51. 4.1.5, 4.9, 4.10
- Carpendale, M. S. T., Cowperthwaite, D. J., Storey, M.-A. D., and Fracchia, F. D. (1997b). Exploring distinct aspects of the distortion viewing paradigm. Technical Report 97-08, School of Computing Science, Simon Fraser University, Burnaby, BC, Canada. 4.1.3
- Chi, E. H. (2000). A taxonomy of visualization techniques using the data state reference model. In *IEEE Symposium on Information Visualization (InfoVis 2000)*, pages 69–75. IEEE Computer Society Press. 4.5.3, 4.52
- Chi, E. H. and Riedl, J. T. (1998). An operator framework for visualization systems. In *IEEE Symposium on Information Visualization (InfoVis 1998)*, pages 63–70. IEEE Computer Society Press. 4.5.3
- Chumakov, I. M. et al. (1995). A YAC contig map of the human genome. *Nature*, 377(Suppl.):175–298. 2
- Cohen, D., Chumakov, I., and Weissenbach, J. (1993). A first generation physical map of the human genome. *Nature*, 336:698–701. 2
- Conklin, J. and Begeman, M. L. (1988). gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, 6(4):303–331. 4.2.1, 4.16, 4.17, 9.3.2
- Coutaz, J. (1990). *Interfaces homme-ordinateur: Conception et réalisation*. Dunod informatique, Paris. 6.2.1
- Coutaz, J. (1993). Software architecture modeling for user interfaces. Technical report, Amodeus Project. Document SM/WP33. An early version of Coutaz (1994). 6.2.1, 6.2.1, 6.2.2
- Coutaz, J. (1994). Software architecture modeling for user interfaces. In Marciniak, J. J., editor, *The Encyclopedia of Software Engineering*, pages 38–49. John Wiley & Sons.
- Cox, D. A., Chugh, J. S., Gutwin, C., and Greenberg, S. (1998). The usability of transparent overview layers. In *SIGCHI '98 Summary*, pages 301–302, Los Angeles CA, USA. ACM Press. 4.3.7, 4.40, 4.3.7, 4.41, 4.3.7, 9.3.2, 9.3.2
- Dib, C., Fauré, S., Fizames, C., Samson, D., Drouot, N., Vignal, A., Millasseau, P., Marc, S., Hazan, J., Seboun, E., Lathrop, M., Gyapay, G., Morissette, J., and

- Weissenbach, J. (1996). A comprehensive genetic map of the human genome based on 5,264 microsatellites. *Nature*, 380(6570):152–154. 1
- Fekete, J.-D. and Plaisant, C. (1999). Excentric labeling: Dynamic neighborhood labeling for data visualization. In *CHI '99*, pages 512–519, Pittsburgh PA, USA. ACM Press. 4.3.8, 9.3.2
- Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47:381–391. 2.2.3
- Fitts, P. M. and Peterson, J. R. (1964). Information capacity of discrete motor responses. *Journal of Experimental Psychology*, 67:103–112. 2.2.3
- Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. (1996). *Computer Graphics: principles and practice*. Addison-Wesley, second in c edition. 2.2
- Fox, D. (1998). *Tabula Rasa: A Multi-scale User Interface System*. PhD thesis, New York University, New York, USA. 6.4.2
- Furnas, G. W. (1986). Generalized fisheye views. In *CHI '86*, pages 16–23, Boston MA, USA. ACM Press. 2.1.7, 4.1.1, 4.1.1, 4.1, 9.3.1, 9.14, 9.3.1, 9.3.2
- Furnas, G. W. and Bederson, B. B. (1995). Space-scale diagrams: Understanding multiscale interfaces. In *CHI '95*, pages 234–241, Denver CO, USA. ACM Press. 4.4, 4.4.1, 4.44, 4.45, 4.46, 9.3.1, 9.3.3
- Furnas, G. W. and Rauch, S. J. (1998). Considerations for information environments and the NaviQue workspace. In *DL '98. Proceedings of the third ACM Conference on Digital libraries*, pages 79–88, Pittsburgh PA, USA. ACM Press. 4.4.4
- Furnas, G. W. and Zacks, J. (1994). Multitrees: enriching and reusing hierarchical structure. In *CHI '94*, pages 330–336, Boston MA, USA. ACM Press. 4.2.6, 4.25, 4.26
- Guimbretière, F. and Winograd, T. (2000). FlowMenu: Combining command, text, and data entry. In *UIST '00*, pages 213–216, San Diego CA, USA. ACM Press. 2.1.6, 2.5, 2.6, 9.12, 9.13
- Harrison, B. L., Ishii, H., Vicente, K. J., and Buxton, W. A. S. (1995a). Transparent layered user interfaces: An evaluation of a display design to enhance focused and divided attention. In *CHI '95*, pages 317–324, Denver CO, USA. ACM Press. 4.3.1, 9.3.2

- Harrison, B. L., Kurtenbach, G., and Vicente, K. J. (1995b). An experimental evaluation of transparent user interface tools and information content. In *UIST '95*, pages 81–90, Pittsburgh PA, USA. ACM Press. 4.3.2, 4.28, 5.2.3, 9.3.2
- Harrison, B. L. and Vicente, K. J. (1996). An experimental evaluation of transparent menu usage. In *CHI '96*, pages 391–398, Vancouver, B.C., Canada. ACM Press. 4.3.1, 4.27
- Herman, I., Melançon, G., and Marshall, M. S. (2000). Graph visualization and navigation in information visualisation: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43. 4.5.2, 4.5.7, 4.5.7, 9.3.1, 9.3.1
- Hopkins, D. (1991). The design and implementation of pie menus. *Dr. Dobb's Journal of Software Tools*, 16(12):16–26,94. 2.1.3, 9.2.1
- Hudson, T. J. et al. (1995). An STS-based map of the human genome. *Science*, 270:1945–1954. 2
- Jacob, R. J. K. and Sibert, L. E. (1992). The perceptual structure of multidimensional input device selection. In *CHI '92*, pages 211–218, Monterey CA, USA. ACM Press. 3.2.2, 9.2.2
- Jin, L. and Banks, D. C. (1997). TennisViewer: A browser for competition trees. *IEEE Computer Graphics and Applications*, 17(4):63–65. 4.2.4
- Johnson, B. and Shneiderman, B. (1991). Tree-maps: A space filling approach to the visualization of hierarchical information structures. In *VIS '91*, pages 284–291. IEEE Computer Society Press. 4.22, 4.2.4
- Jul, S. and Furnas, G. W. (1998). Critical zones in desert fog: Aids to multiscale navigation. In *UIST '98*, pages 97–106, San Francisco, CA, USA. ACM Press. 4.4.8
- Keahey, T. A. and Robertson, E. L. (1996a). Non-linear image magnification. Technical Report 460, Department of Computer Science, Indiana University, USA. 4.13, 4.14, 4.5.5
- Keahey, T. A. and Robertson, E. L. (1996b). Techniques for non-linear magnification transformations. In *IEEE Symposium on Information Visualization (InfoVis 1996)*, pages 38–45. IEEE Computer Society Press. 4.1.8, 4.1.8
- Kernighan, B. W. and Ritchie, D. M. (1989). *The C Programming Language*. Prentice Hall, 2 edition. 4.1.1

- Kramer, A. (1994). Translucent patches—dissolving windows. In *UIST '94*, pages 121–130, Marina del Rey CA, USA. ACM Press. 4.3.4, 4.30, 4.31
- Kramer, A. (1996). Translucent patches. *Journal of Visual Languages and Computing*, 7(1):57–77. 4.3.4
- Kurtenbach, G., , Fitzmaurice, G. W., Owen, R. N., and Baudel, T. (1999). The hotbox: efficient access to a large number of menu-items. In *CHI '99*, pages 231–237, Pittsburgh PA, USA. ACM Press. 2.1.4
- Kurtenbach, G. (1993). *The Design and Evaluation of Marking Menus*. PhD thesis, University of Toronto. 2.1.5, 2.2, 2.13, 2.14, 3.1.3
- Kurtenbach, G. and Buxton, W. (1993). The limits of expert performance using hierarchic marking menus. In *CHI '93*, pages 482–487, Amsterdam, The Netherlands. ACM Press. 2.1.4, 2.3, 9.2
- Kurtenbach, G. and Buxton, W. (1994). User learning and performance with marking menus. In *CHI '94*, pages 258–264, Boston MA, USA. ACM Press. 2.1.4, 9.2.1, 9.2.2
- Kurtenbach, G., Fitzmaurice, G., Baudel, T., and Buxton, B. (1997). The design of a GUI paradigm based on tablets, two-hands, and transparency. In *CHI '97*, pages 35–42, Atlanta GA, USA. ACM Press. 4.3.1, 4.3.1
- Lamping, J. and Rao, R. (1994). Laying out and visualizing large trees using a hyperbolic space. In *UIST '94*, pages 13–14, Marina del Rey CA, USA. ACM Press. 4.1.7
- Lamping, J. and Rao, R. (1996). The hyperbolic browser: A focus+context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, 7(1):33–35. 4.1.7, 4.1.7
- Lamping, J., Rao, R., and Pirolli, P. (1995). A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *CHI '95*, pages 401–408, Denver CO, USA. ACM Press. 4.1.7, 4.12
- Lecolinet, E., Fekete, J.-D., and Pook, S. (2001). Bibliothèques : comparaisons entre le réel et le virtuel en 3D, 2D zoomable et 2D arborescent. In *ASTI 2001*, pages 24–25. Association Française des Sciences et Technologies de l'Information. 7.2, 9.3.4
- Leung, Y. K. and Apperley, M. D. (1994). A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, 1(2):126–160. 4.5.1, 4.51

- Lieberman, H. (1994). Powers of ten thousand: Navigating in large information spaces. In *UIST '94*, pages 15–16, Marina del Rey CA, USA. ACM Press. 4.4.6
- Lieberman, H. (1997). A multi-scale, multi-layer, translucent virtual space. In *IEEE Symposium on Information Visualization (InfoVis 1997)*, pages 124–131. IEEE Computer Society Press. 4.4.6, 4.50
- Macedonia, M. (2000). Entertainment computing: Using technology and innovation to simulate daily life. *Computer*, 33(4):110–112. 2.1.3
- MacKenzie, I. S. (1995). Movement time prediction in human-computer interfaces. In Baecker, R. M., Buxton, W. A. S., Grudin, J., and Greenberg, S., editors, *Readings in human-computer interaction*, pages 483–493. Kaufmann, second edition. 9.2.1
- Mackinlay, J. D., Robertson, G. G., and Card, S. K. (1991). The perspective wall: Detail and context smoothly integrated. In *CHI '91*, pages 173–176, New Orleans LA, USA. ACM Press. 4.1.2, 4.1.2, 4.1.3
- Myers, B. A., McDaniel, R. G., and Miller, R. C. (2000). The Amulet prototype-instance framework. In Fayad, M. and Johnson, R. E., editors, *Domain-Specific Application Frameworks*, pages 529–546. John Wiley & Sons. 6.2.1
- Nielson, J. (1987). Classification of dialog techniques. *ACM SIGCHI Bulletin*, 19(2):30–35. 5.2.7
- Nigay, L. and Coutaz, J. (1992). PAC-Expert: Towards an automatic generation of dialogue controllers. Technical report, LGI-IMAG, University of Grenoble, BP 53X, 38041 Grenoble cedex, France. also published as Document D18, Project BRA Amodeus 3066. 6.2.1
- Nigay, L. and Coutaz, J. (1995). A generic platform for addressing the multimodal challenge. In *CHI '95*, pages 98–105, Denver CO, USA. ACM Press. 6.2.1
- Noik, E. G. (1993). Exploring large hyperdocuments: fisheye views of nested networks. In *Hypertext '93*, pages 192–205, Seattle WA, USA. ACM Press. 4.1.1
- Olsen, Jr., D. R. (1998). *Developing User Interfaces*. Morgan Kaufmann. 6.2.1
- Ousterhout, J. K. (1994). *Tcl and the Tk Toolkit*. Addison-Wesley. 6.4.3
- Perkins, Jr., E. C. (1993). Bedlam in Simcity. *UnixWorld*. 2.2.2

- Perlin, K. (1998). Quikwriting: continuous stylus-based text entry. In *UIST '98*, pages 215–216, San Francisco, CA, USA. ACM Press. 2.1.6
- Perlin, K. and Fox, D. (1993). Pad: An alternative approach to the computer interface. In *SIGGRAPH '93*, pages 57–64, Anaheim CA, USA. ACM Press. 6.4.1
- Pfaff, G. E., editor (1985). *User Interface Management Systems*. Springer-Verlag. Proceedings of the IFIP/EG Workshop on User Interface Management Systems, Seeheim, Fed. Rep. Germany, Oct. 1983. 6.2.1
- Plaisant, C., Shneiderman, B., Doan, K., and Bruns, T. (1999). Interface and data architecture for query preview in networked information systems. *ACM Transactions on Office Information Systems*, 17(3):320–341. 4.49, 4.4.5
- Rao, R. and Card, S. K. (1994). The Table Lens: Merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In *CHI '94*, pages 318–322, 481–482, Boston MA, USA. ACM Press. 4.1.6, 9.3.1
- Rao, R. and Card, S. K. (1995). Exploring large tables with the table lens. In *CHI '95 (conference companion)*, pages 403–404, Denver CO, USA. ACM Press. 4.1.6, 9.3.1
- Raskin, J. (2000). *The Humane Interface*. Addison-Wesley. 2.1.2, 2.2.11, 3.1.9, 3.5.2, 5.2.7, 9.2.1
- Reinhardt, A. (1991). First impressions: Momenta points to the future. *Byte Magazine*. 2.1.5
- Resnikoff, H. L. (1989). *The Illusion of Reality*. Springer, New York, first edition. 4.1.2
- Robert, L. and Lecolinet, É. (1998). Browsing hyperdocuments with multiple focus+context views. In *ACM Conference on Hypertext and Hypermedia (HT'98)*. 4.2.3
- Robert, L. and Lecolinet, É. (2001). Digital annotation and exploration techniques for handling image-based hypermedia. In *IFIP TC.13 Conference on Human-Computer Interaction (INTERACT 2001)*. 4.2.3
- Robertson, G. G., Card, S. K., and Mackinlay, J. D. (1993). Information visualization using 3D interactive animation. *Communications of the ACM*, 36(4):56–71. 4.2.5

- Robertson, G. G. and Mackinlay, J. D. (1993). The document lens. In *UIST '93*, pages 101–108, Atlanta GA, USA. ACM Press. 4.3, 4.1.3, 9.3.2
- Robertson, G. G., Mackinlay, J. D., and Card, S. K. (1991). Cone trees: Animated 3D visualizations of hierarchical information. In *CHI '91*, pages 189–194, New Orleans LA, USA. ACM Press. 4.23, 4.2.5
- Robinson, A. J. and Flores, T. P. (1997). Novel techniques for visualising biological information. In *ISMB 97*, pages 241–249. AAAI Press, Menlo Park, California. 4.3.6
- Sarkar, M., Snibbe, S. S., Tversky, O. J., and Reiss, S. P. (1993). Stretching the rubber sheet: A metaphor for viewing large layouts on small screens. In *UIST '93*, pages 81–91, Atlanta GA, USA. ACM Press. 4.4, 4.1.4, 4.5
- Schaffer, D., Zuo, Z., Greenberg, S., Bartram, L., Dill, J., Dubs, S., and Roseman, M. (1996). Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Transactions on Computer-Human Interaction*, 3(2):162–188. 4.2.2, 4.19, 4.20
- Scheifler, R. W., Gettys, J., and Rosenthal, D. (1992). *X Window System: The Complete Reference to Xlib, X Protocol, Iccm, Xlfd*. Digital Press, third edition. 6.4.3
- Sheffield, V. C. et al. (1995). A collection of tri- and tetranucleotide repeat markers used to generate high quality, high resolution human genome-wide linkage maps. *Human Molecular Genetics*, 4(10):1837–1844. 1
- Shneiderman, B. (1983). Direct manipulation: A step beyond programming languages. *IEEE Computer*, 16(8):57–69. 4.5.4
- Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. In *VL '96*, pages 336–343. IEEE Computer Society Press. 4.5.2, 4.5.2, 4.5.2
- Silvers, R. (1995). Livemap—a system for viewing multiple transparent and time-varying planes in a three dimensional space. In *CHI '95 (conference companion)*, pages 200–201, Denver CO, USA. ACM Press. 4.3.1
- Stone, M. C., Fishkin, K., and Bier, E. A. (1994). The movable filter as a user interface tool. In *CHI '94*, pages 306–312, Boston MA, USA. ACM Press. 4.3.6, 4.3.6, 4.35, 4.37
- Stroustrup, B. (1997). *The C++ Programming Language*. Addison-Wesley. 6.3.1

- The Open Group (1997). *Motif 2.1 – Programmer's Reference*. The Open Group. 6.2.1
- Tufte, E. R. (1998). *Visual Explanations: Images and Quantities, Evidence and Narrative*. Graphic Press. 4.4.5
- Turo, D. and Johnson, B. (1992). Improving the visualization of hierarchies with treemaps: Design issues and experimentation. Technical Report CS-TR-2901, Human-Computer Interaction Laboratory, Department of Computer Science, University of Maryland. also in Proceedings of IEEE Visualization '92. 4.2.4
- Tweedie, L. (1997). Characterizing interactive externalizations. In *CHI '97*, pages 375–382, Atlanta GA, USA. ACM Press. 4.53, 4.5.4, 4.54, 4.5.4, 4.5.4, 4.5.4
- Vernier, F. (2001). *Output multimodality: A case study of visualization techniques for large information spaces*. PhD thesis, Université Joseph Fournier, Grenoble, France. 4.5.5, 4.5.5, 4.5.5, 4.5.5
- Viara, E., Barillot, E., and Vaysseix, G. (1999). The EyeDB OODBMS. In *IDEAS'99 International Database Engineering and Applications Symposium*, pages 390–402, Montreal, Canada. IEEE publications. 7.1.2, 9.3.3
- Viega, J., Conway, M. J., Williams, G., and Pausch, R. (1996). 3D magic lenses. In *UIST '96. Proceedings of the ACM symposium on User interface software and technology*, pages 51–58, Seattle WA, USA. ACM Press. 4.38, 4.3.6, 4.39
- Woodruff, A., Landay, J., and Stonebraker, M. (1998a). Constant information density in zoomable interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces AVI '98*, pages 57–65, L'Aquila, Italy. ACM Press. 4.4.5, 4.4.5, 4.47, 4.48
- Woodruff, A., Landay, J., and Stonebraker, M. (1998b). Goal-directed zoom. In *SIGCHI '98 Summary*, pages 305–306, Los Angeles CA, USA. ACM Press. 4.4.7
- Zhai, S., Buxton, W., and Milgram, P. (1994). The “Silk Cursor”: Investigating transparency for 3D target acquisition. In *CHI '94*, pages 459–464, Boston MA, USA. ACM Press. 4.3.3, 4.29

Index

- 3D pliable surface, 83, 88, 89, 127, 128, 136, 141
- Acrobat Reader, 48, 49
- Amulet, 174
- applet, 164, 179
- Arch, 173, 175
- AWT, 180, 181, 183
- C++, 176, 178, 184
- Cam Tree, 97
- CDI, 175, 198
- chunk, 48
- Command Compass, 31, 46
- Cone Tree, 96, 98, 126, 134, 138
- constant information density, 120
- context layer, 149
- critical zones, 125, 148
- Data State Model, 130
- DataSplash, 120
- desert fog, 123
- design patterns, 134
- direct manipulation, 133
- Document Lens, 80, 128, 140, 141
- double buffering, 181
- dynamic queries, 122, 129
- Excentric Labelling, 113, 115, 147
- EyeDB, 191
- fisheye menu, 34
- fisheye view, 76, 77, 88, 92, 97, 127, 128, 136, 138, 141
- Fitts' Law, 38, 67, 68
- FlowMenu, 32–34, 46, 56–59
- gIBIS, 90, 98, 148
- goal-directed zoom, 123
- history layer, 158
- Hotbox, 31, 54
- HTML, 185
- HuGeMap, 175, 189–191
- hyperbolic display, 86, 89, 139, 141, 154
- indirect manipulation, 133
- Information Visualizer, 97
- integral parameters, 62, 68
- Internet Explorer, 179
- Java, 164, 176, 179, 180, 183, 186
- Jazz, 186
- JDK, 180
- lazy evaluation, 167
- lost in hyperspace, 125
- Macroscope, 122, 127, 128, 138
- Magic Lens, 69, 95, 100, 106, 108, 115, 116, 134, 161, 162, 169, 192, 202
- magnifying glass, 108
- marking menu, 27, 30, 32, 41, 45, 48, 53, 56–59, 72, 102
- Momenta, 31
- Multitree, 98
- MVC, 173, 174
- NaviQue, 119

- Netscape, 28, 43, 44, 179
- non-linear transformations, 88, 127
- OpenGL, 183, 187
- orthogonal stretching, 81, 82, 84
- PAC, 174, 177
- PAC-Amodeus, 174
- Pad, 184
- Pad++, 118, 120, 184, 186
- Perspective Wall, 77, 127, 135, 141
- pie menu, 27, 29, 31, 38, 53, 57–59, 72
- polygonal stretching, 82
- pop-up menu, 5, 27, 29, 38, 45, 48, 54, 57, 202
- quasimodal, 29, 46, 155
- Quikwriting, 34
- radial menu, 29, 30
- residue, 125
- Rubber Sheet, 81, 127, 128, 139, 141
- scene graph, 187
- Seeheim, 172, 173
- SimCity, 38
- Slinky, 173, 174
- spring-loaded modes, 155
- Table Lens, 84, 127, 134, 139, 141
- Tabula Rasa, 184
- Tcl/Tk, 185
- TennisViewer, 95
- The Sims, 30
- thin client, 180
- Tivoli, 41
- tool tip, 113, 147
- Toolglass, 100, 102, 105, 109, 115
- Translucent Patches, 103
- Transparent Overview Layers, 111, 127, 128, 138
- Treemap, 95, 98, 126
- triglyph, 165
- virtual library, 198
- visual information seeking mantra, 128
- volumetric lens, 110
- vreng, 69
- Wallace and Gromit, 161
- Write Once, Run Anywhere, 180
- X Window System, 184
- Xerox Star, 36, 44
- ZoomTree, 94, 98