N° d'ordre: 4651

THÈSE

présentée à

l'Université Louis Pasteur
Laboratoire des Sciences de l'Image,
de l'Informatique et de la Télédétection
UMR 7005 CNRS-ULP

par

Mlle Dominique GERBER

pour obtenir le grade de

Docteur de l'université Louis Pasteur de Strasbourg 1
Mention SCIENCES
Spécialité INFORMATIQUE

Interaction 3D sur un plan de travail virtuel : Application aux déformations de forme libre

---

soutenue publiquement le **25 Octobre 2004**,
devant la commission d'examen composée de :

Mme. Dominique BECHMANN, Directeur de Thèse,
Professeur à l'Université Louis Pasteur de Strasbourg
M. Pascal SCHRECK Rapporteur Interne,
Professeur à l'Université Louis Pasteur de Strasbourg
Mme Sabine COQUILLART Rapporteur Externe,
Directeur de Recherches INRIA
M. Bernard PEROCHE Rapporteur Externe,
Professeur à l'Université Claude Bernard de Lyon
M. Philippe FUCHS Examinateur,
Professeur à l'école des mines de Paris

Remerciements

Je remercie toutes les personnes que j'ai côtoyées durant ces trois années de thèse dans mon laboratoire d'accueil, le LSIIT, ainsi qu'au département d'informatique de l'Université Louis Pasteur, qui se sont toutes révélées être d'excellents collègues avec qui j'ai apprécié de passer du temps.

Je remercie en particulier Mme Bechmann, ma directrice, pour nos nombreuses discussions fort intéressantes, ses conseils toujours utiles et ses nombreuses relectures de mes travaux.

Je remercie Mme Coquillart, M. Fuchs, M. Péroche et M. Schreck et pour m'avoir fait l'honneur de juger mon travail.

Je tiens également à remercier Olivier Génevaux, avec qui j'ai eu nombre de discussions constructives sur mes travaux. Je remercie également Ludovic Sternberger, pour m'avoir fourni de précieux renseignements durant la phase de rédaction de cette thèse.

Enfin, je remercie toute ma famille et mes amis, qui ont su me supporter durant toutes ces années d'études et m'ont soutenue de leurs encouragements lorsque j'en avais besoin.

Table des matières

Introduction

«Le message venait d'arriver, gravé dans une boule de bois vernie de la taille d'une balle de golf : *Sarah Marks*, *Donald Dubin*. Une bille de merisier, rouge : un double homicide passionnel allait être commis. John Anderton, Préfet de Police à l'organisation Précrime, savait qu'il devait faire vite : l'acte irréparable allait être commis dans moins de 15 minutes. Ce qu'il ignorait, c'était l'endroit où serait commis cet acte. Il n'avait que peu de temps et savait par expérience que tout allait se précipiter à partir du moment où il se serait connecté sur le système de centralisation des données de l'Organisation, ce qu'il fit machinalement. Il n'était que 8h04 du matin, un jeudi ; la journée commençait avec l'affaire 11 – 08.

John Anderton se connecta au système de centralisation et revêtit les interfaces sensorielles sur chacune de ses mains. Quelques projecteurs holographiques s'allumèrent de concert, formant autour de lui une procession de panneaux sur lesquels défilaient des vidéos d'un homme, le meurtrier. C'étaient les visions chaotiques des *précogs*, ces êtres moitié mutants qui pouvaient prédire l'avenir à court terme, concernant l'affaire 11 – 08. Anderton devait, à partir de toutes ces pièces éparses, déduire le lieu où le forfait allait être commis, afin d'y envoyer ses unités et d'empêcher le criminel de nuire.

Il saisit un premier panneau, le plaça devant lui et le laissa défiler. L'extrait ne durait que quelques secondes. Aucun intérêt pour l'instant, il le mit de côté de sa main gauche, pendant que la main droite désignait un autre panneau, qui prit la place du précédent qui retournait à son emplacement d'origine. Une maison, en face d'un petit parc. C'est tout ce que cette séquence lui apprit. Trop vague pour l'instant. Ses mains partirent à la recherche d'autres informations, dissimulées dans les séquences embrouillées captées par les précogs. Vu de l'extérieur, on aurait pu croire qu'Anderton exécutait une chorégraphie étrange et silencieuse. Ses mains et ses doigts cherchaient, au milieu de nulle part, des informations que seul lui-même pouvait voir. Parfois, Anderton avait besoin d'une donnée qui ne figurait pas dans son environnement ; il faisait alors appel à l'un de ses collaborateurs, assis en retrait dans la salle, qui chargeait la donnée manquante.

Le temps s'écoulait, et l'échéance approchait. Pourtant Anderton n'avait pas encore trouvé l'endroit où le crime serait commis. Il s'agita de plus belle, dans son ballet silencieux : mouvement latéral pour faire défiler les séquences, main levée

et geste vers l'avant pour placer un panneau en arrière plan... Le nombre de combinaisons gestuelles était impressionnant, et permettait à Anderton, qui manipulait le système depuis près de trente ans, d'atteindre des performances impressionnantes. La vision et la coordination des gestes étaient utilisées à leur maximum...»

Cette scène, transcrite du film *Minority Report*[1], illustre la réalité virtuelle telle que nous pouvons actuellement l'imaginer : efficace, agréable, intuitive. Bref, parfaite. Pourtant, si le cinéma nous en offre une vision idyllique, la réalité nous montre que bien des obstacles restent à franchir avant d'arriver à rejoindre la fiction, malgré des avancées fulgurantes faites depuis les premiers balbutiements de la discipline : affichage, interaction, fatigue, etc.

En fait, l'homme cherche depuis toujours à imiter, copier ou modifier la réalité. Les raisons en sont diverses. L'art par exemple, est une forme de réalité virtuelle dans laquelle l'artiste créé une vision qui lui est propre d'un endroit, d'une situation ou d'une personne, par l'intermédiaire d'une peinture ou d'une photographie. Les trompe-l'œil sont un excellent exemple, qui, en usant d'illusions d'optique font croire à une porte là où il n'y a qu'un mur. La sculpture également permet au sculpteur de créer un objet selon ses propres désirs. Les drogues bien entendu, en altérant nos facultés perceptives et sensorielles, plongent le consommateur dans une sorte de réalité décalée, qui n'existe que pour le lui. Plus récemment, les progrès fulgurants de l'informatique ont rendue possible l'existance de mondes virtuels interactifs en images de synthèse, donc il sera question tout au long de cet ouvrage.

Réalité virtuelle. Ce nom, volontairement contradictoire, cache beaucoup de choses. Pour certains, la réalité virtuelle doit remplacer tous nos repères habituels. Pour d'autres, elle doit simplement améliorer ce qui *pose problème* où est limitant dans le monde réel. La réalité virtuelle prend une place de plus en plus importante de nos jours : l'industrie automobile y voit un moyen de faire du prototypage virtuel de véhicules à moindre frais, les musées y trouvent un moyen de faire visiter des lieux ancestraux disparus, les chimistes peuvent explorer les molécules comme jamais auparavant, les psychiatres y voient un moyen de soigner diverses phobies, etc.

Cependant, même si son avenir semble prometteur, la réalité virtuelle est une discipline très récente, et souffre à ce titre de nombreuses faiblesses. Sans parler du matériel souvent cher et encombrant, les environnements de réalité virtuelle restent peu nombreux : on ne dispose pas aujourd'hui d'environnements de réalité virtulle «grand public», comme c'est le cas pour les ordinateurs personnels que l'on trouve dans bon nombre de foyers. De plus, le passage à la troisième dimension pour l'affichage, la manipulation où l'interaction avec les programmes informatiques pose de nombreux problèmes auxquels aucune solution définitivement satisfaisante n'a été proposée.

Nous verrons, au cours de ce manuscrit, un prototype d'interaction 3D, ayant

[1] Film réalisé en 2003 par Steven Spielberg, d'après une œuvre de Phil K. Dick

pour cadre la modélisation géométrique, et plus particulièrement les déformations de forme libre. Ce prototype servira de base à l'élaboration d'une technique de menu innovante.

Ce manuscrit de thèse se décompose en quatre parties. Tout d'abord, nous effectuerons un tour d'horizon de la réalité virtuelle aujourd'hui, en nous concentrant particulièrement sur les techniques logicielles d'interaction. Nous ferons également un rapide tour d'horizon des déformations de forme libre et notamment du modèle de déformation DOGME sur lequel nous appuierons notre prototype expérimental, qui fera l'objet de la seconde partie. Nous y aborderons les principales manipulations et interactions qu'il permet, ainsi que les différentes techniques mises en oeuvre. Nous verrons dans une troisième partie l'élaboration d'une métaphore de menu, en réponse à des manques et des problèmes rencontrés dans la partie précédente. Enfin, nous terminerons par quelques détails d'implémentation propres à la réalité virtuelle auxquels il a fallu faire face et pour lesquels nous avons proposé des solutions.

Première partie

État de l’art

Chapitre 1

La réalité virtuelle

1.1 Introduction

Le terme de *réalité virtuelle* est de lui-même contradictoire. La réalité est, par définition, ce qui existe, alors que le virtuel est, par définition, ce qui n'existe pas. Que représente alors cette «réalité virtuelle», cet «existant qui n'existe pas» ? Cette question divise la population en plusieurs camps, selon sa propre notion du «virtuel». Si l'on se place dans la réalité habituelle, celle dans laquelle nous vivons, nous pouvons y adjoindre des éléments «virtuels» par le biais de la technologie. Ces adjonctions forment alors une sorte de continuum, dans lequel on passe graduellement de la réalité «pure», celle où tout est «réel», à la virtualité «pure», celle où tout est «virtuel». La figure 1.1 illustre ce continuum, avec les principales techniques à ce jour selon le degré de réel et de virtuel qu'elles mélangent.

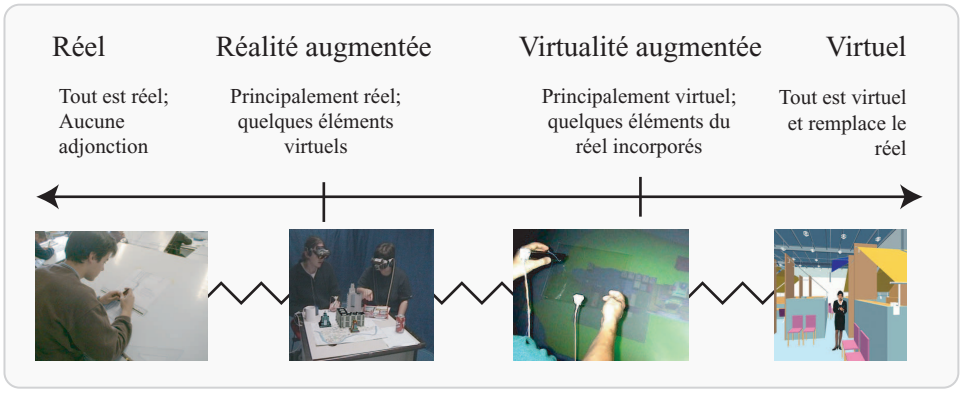


FIG. 1.1 – Du réel au virtuel : un continuum.

Nous considérerons dans la suite de cette thèse la réalité virtuelle comme étant celle proche de la virtualité pure, ou de la virtualité augmentée. Dans cette situation, l'un des enjeux majeurs est de procurer à l'utilisateur un sentiment d'*immersion* important. Nous allons dans un premier temps introduire plus précisé-

ment ce que sous-entend la notion de réalité virtuelle. Nous verrons ensuite les principaux périphériques matériels utilisés actuellement. Enfin, nous verrons les techniques logicielles mises en œuvre pour effectuer diverses tâches, comme se déplacer, sélectionner, manipuler et contrôler.

1.1.1 Immersion

L'un des grands attraits de la réalité virtuelle réside dans le fait qu'elle puisse procurer à l'utilisateur le sentiment de faire partie du monde virtuel ou des données qui s'y présentent, comme s'il y était *pour de vrai*. Ce sentiment est nommé immersion. On peut en fait distinguer deux types d'immersions : l'immersion cognitive et l'immersion sensorimotrice.

L'immersion cognitive caractérise la conviction que l'utilisateur a «d'y être». Cette notion n'est d'ailleurs pas spécifique à la réalité virtuelle. En effet, une excellente immersion cognitive peut être atteinte à l'aide d'un roman, par exemple. Cependant, une bonne immersion cognitive est toujours recherchée, car elle permet à l'utilisateur d'être totalement concentré sur sa tâche.

L'immersion sensorimotrice caractérise la sensation que l'utilisateur a «d'y être». Cette notion est plus spécifique à la réalité virtuelle. Cette immersion est atteinte en trompant un ou plusieurs sens de l'utilisateur, par leur stimulation artificielle mais cohérente par rapport à l'environnement virtuel environnant. On peut noter que la réalité virtuelle n'est pas la seule discipline cherchant à maximiser l'immersion sensorimotrice. A de comparaison, le cinéma procure une immersion sensorimotrice de la vue et de l'ouïe. Cependant, l'immersion sensorimotrice en réalité virtuelle est plus poussée, puisqu'elle prend en compte davantage d'informations issues de l'utilisateur, comme sa position pour actualiser la scène en conséquence de ses déplacements par exemple. Parmi tous les sens de l'utilisateur, la vue tient une place particulière lors d'une simulation de réalité virtuelle : c'est par la vision que tout commence. C'est donc ce sens que l'on cherchera à tromper en premier lieu et le plus efficacement possible. Mais ce n'est pas le seul. L'ouïe, par exemple, peut apporter un complément d'information significatif. Le toucher également, s'avère être dans certains cas une source d'information primordiale. A l'heure actuelle, goût et odorat restent un peu à l'écart des environnements virtuels, souvent inodores.

Stimuler efficacement la vision, sur un environnement semi-immersif, passe presque toujours par un affichage stéréoscopique, permettant au cerveau de recréer la profondeur grâce à un processus de fusion d'images. Pour tirer pleinement profit de la vision binoculaire, l'ensemble de la scène doit suivre les mouvements de l'utilisateur, de sorte qu'il puisse naturellement regarder autour d'un objet, au dessus, etc. Le suivi du mouvement de la tête, duquel on déduit la position des yeux, est utilisé pour cela. Nous détaillerons les périphériques utilisés pour la vision et le suivi du mouvement dans la section 1.2.

Plusieurs études sur ce sentiment d'immersion sensorimotrice en milieu virtuel ont été conduites. Meehan [MRWJ03] s'est penché sur le problème de la fréquence

d'affichage des images et leur effet sur le sentiment d'immersion. Il ne note pas d'effet désastreux d'un faible taux de rafraîchissement sur l'immersion. [LDP+02] s'est intéressé à l'importance du champ de vision, et note que plus le champ visuel est couvert par la scène virtuelle, meilleure est l'immersion. LaViola [LaV00] s'est quant à lui intéressé au *mal du virtuel*, une sensation de mal de mer qui peut survenir lors d'une utilisation plus ou moins prolongée d'un environnement virtuel[1]

1.1.2 Entre matériel et logiciel

Dans ce cadre, le terme de réalité virtuelle sous-entend deux choses : un matériel spécifique et des techniques logicielles adaptées à ce matériel. Parmi les périphériques non conventionnels, on compte tout particulièrement les dispositifs d'affichage non conventionnels (casque, grand écran, écran sphérique, etc.) affichant des images en stéréo, et les périphériques de suivi de mouvement. On associe d'ailleurs souvent la réalité virtuelle à ces deux seuls périphériques. Il s'agit là d'une erreur commune, mais la réalité est bien différente. Comme tout environnement informatique interactif, un dispositif d'affichage représente la base du système. A ceci s'ajoute le ou les périphériques d'entrée, que l'utilisateur pourra utiliser pour dialoguer avec le système. La classique souris est en général remplacée par un périphérique à six degrés de libertés –trois rotations, trois translations. Charge aux développeurs de savoir tirer profit de ce matériel et des possibilités qu'il offre, car le matériel ne résout pas les problèmes mais donne des clés pour les résoudre en imaginant des techniques d'interaction innovantes, que l'on appelle *paradigme d'interaction*.

Définition Un paradigme d'interaction est un ensemble de règles et de techniques permettant à l'utilisateur d'effectuer une tâche au sein d'un système interactif en lui offrant un parallèle entre la réalité et le monde virtuel.

Un système de réalité virtuelle est donc défini d'une part par sa spécificité matérielle, mais aussi et surtout par l'ensemble de techniques logicielles mises en oeuvre. Nous verrons en effet que la partie logicielle est souvent, du moins dans une certaine mesure, indépendante du matériel lui-même, et peut s'adapter à diverses configurations. L'enjeu est alors de choisir les techniques les plus performantes, tout en tenant compte du matériel dont on dispose, afin de tirer un maximum du système dont on dispose.

1.2 La réalité matérielle

La réalité virtuelle est rendue possible par des évolutions matérielles. Ce matériel, spécifique ou emprunté à d'autres disciplines, forme un ensemble évoluant

[1] Tout comme le mal de mer, le mal du virtuel est dû à une incohérence entre la vue et ce que perçoit l'oreille interne.

avec les progrès de l'une ou l'autre technique. Si la réalité virtuelle est l'existance d'un monde virtuel, la réalité matérielle est l'existance du matériel qui permet l'existance du virtuel.

Nous allons dans un premier temps faire un rapide tour d'horizon des principaux matériels utilisés en réalité virtuelle. Nous aborderons exclusivement les périphériques d'affichage, de suivi de mouvement et d'entrée. D'autres périphériques sont parfois utilisés, comme les dispositifs de retour d'efforts ou de sonorisation tridimensionnelle, mais ne seront pas abordés ici.

1.2.1 Les débuts

L'idée de réalité virtuelle ne date pas d'hier, même si on ne l'appelait pas ainsi. Sans chercher à remonter trop loin dans le passé, on peut tout de même noter l'appareil conçu en 1838 par Sir Charles Wheatstone, permettant de percevoir le relief grâce à une astucieuse utilisation de miroirs et de clichés. Ce principe fut repris et amélioré quelques années plus tard, en 1844, avec un procédé permettant de voir des photos «stéréo», prise à l'aide d'un appareil à prismes conçu par David Brewster. D'une certaine manière, ce sont ces appareils qui lancèrent la mode des «images en relief». Mais malgré l'engouement que ce type de dispositifs créèrent à leur époque, ils demeuraient des jouets, car ils ne permettaient de visualiser que des images immobiles (photographies, dessins, etc.). Ce n'est qu'avec l'arrivée de l'informatique que ceci allait changer, le cinéma n'ayant pas réellement cherché à tirer profit de ces possibilités.

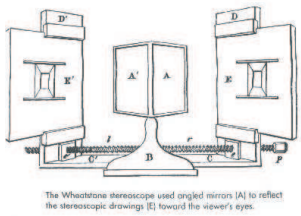


FIG. 1.2 – Le stéréoscope de Sir Wheatstone (1802–1875)

Vers la fin des années 1950, l'informatique faisait son apparition, sous la forme de gigantesques machines à calculer, utilisées par une poignée d'experts parlant d'incompréhensibles dialectes de programmation aujourd'hui tombés dans l'oubli. Ces machines n'étaient alors utilisées que comme des calculatrices géantes, servant à traiter d'interminables listes de chiffres. Douglas Engelbart, un ancien technicien radar de la marine, avait cependant l'intuition que ces machines pourraient également produire des images, que l'on afficherait sur des écrans similaires aux radars qu'il connaissait bien. Rapidement, l'idée fit son chemin et beaucoup de personnes s'y intéressèrent. Dès 1966, Ivan Sutherland créé le premier casque de réalité virtuelle, permettant d'immerger un utilisateur dans un monde créé par un ordinateur. Le système fut amélioré en 1970 par Daniel Vickers par l'adjonction de la stéréoscopie et du suivi de mouvement : La réalité virtuelle *moderne* voyait le jour.

1.2.2 Le présent

Aux yeux du grand public, l'évocation de la réalité virtuelle fait généralement penser une personne portant un casque immersif et des gants de donnée. Cette vi-

sion était peut être exacte dans les années 1980, mais ne l'est plus aujourd'hui, notemment depuis l'arrivée des environnements semi-immersifs qui ont su s'imposer comme nouveau standard d'affichage de la réalité virtuelle dans un grand nombre de laboratoires et d'entreprises.

Affichage

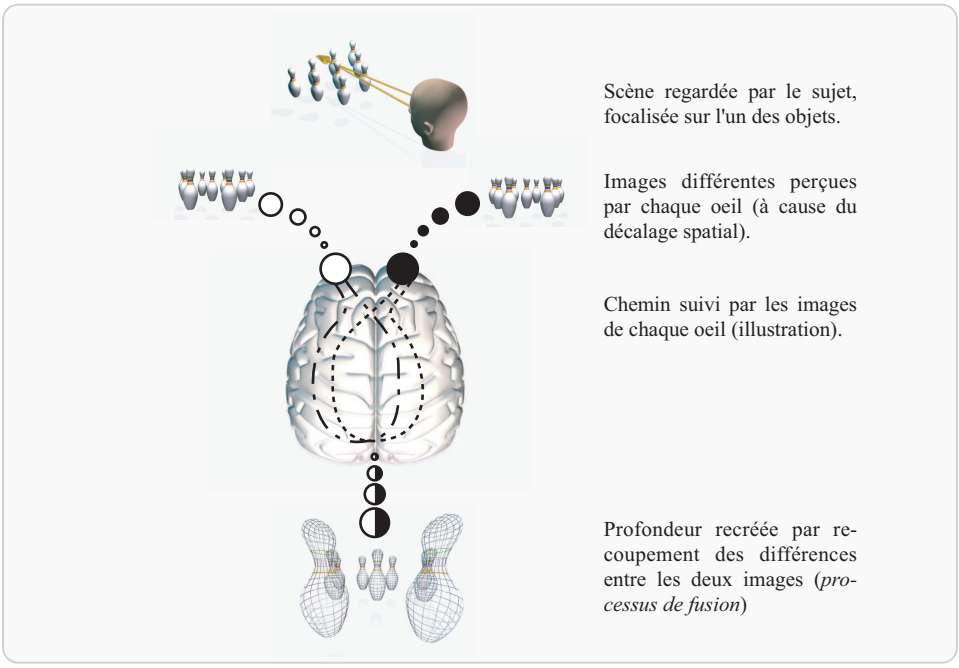
Un élément clé de tout système de réalité virtuelle est son dispositif d'affichage, qui conditionne une grande partie de la sensation d'immersion que l'utilisateur pourra en espérer. Comme chacun sait, de nos cinq sens, la vision est largement celui sur lequel nous reposons le plus, d'où l'importance de maximiser la qualité de l'immersion visuelle. Ceci passe quasiment toujours pas la vision stéréoscopique, permettant au cerveau de reconstituer l'information de profondeur à partir des différences visuelles perçues par nos deux yeux. Un autre facteur déterminant pour l'immersion visuelle est la proportion de notre champ visuel couvert par la scène virtuelle, qui doit évidemment être aussi grande que possible.

Dans nos actions quotidiennes, nous utilisons sans nous en rendre compte une capacité reliée à la vision : la capacité à percevoir la profondeur, le relief. Cette capacité nous aide à déterminer si un objet est proche ou lointain, si notre main qui veut saisir une tasse est à la bonne profondeur, etc. Ce mécanisme de perception de la profondeur est possible grâce à la présence de deux yeux, légèrement décalés. Chacun d'eux reçoit l'image de la même scène, mais légèrement décalée. Ces deux images sont envoyées au cerveau, qui va y appliquer un processus de *fusion*, consistant à exploiter les différences entre les deux images pour extraire l'information de profondeur. La figure 1.3 illustre le principe de la fusion des images. Ce principe d'exploitation de différences dans deux images est d'ailleurs utilisé dans nombre d'applications de reconnaissance de formes, pour la robotique notamment. Ce même principe est également mis en oeuvre dans les systèmes de réalité virtuelle, qui sera chargé de générer une image adaptée à chaque oeil, pour fournir au cerveau les mêmes informations qu'il aurait eues si la scène avait été réelle, pour permettre ce processus de fusion.

Casques Les casques de réalité virtuelle[2] permettent d'occuper une grande partie du champ visuel en plaçant des écrans dotés d'une optique adaptée très près des yeux. On est ainsi totalement immergé dans la scène, puisqu'en le couplant avec un système de suivi de position, on a virtuellement des écrans partout autour de sa tête : où que l'on regarde, la scène virtuelle est présente. L'utilisateur a alors l'impression de voir par les yeux de quelqu'un qui est dans la scène virtuelle.

De nombreux modèles commerciaux ont vu le jour (quelques exemples sont donnés en figure 1.4), mais ne connurent pas un succès phénoménal à cause de deux plusieurs inconvénients. D'une part, la résolution des écrans reste souvent trop faible –la scène virtuelle est trop «pixellisée» pour procurer vision satisfai-

[2] HMD, Head Mounted Display



Chaque oeil capte sa vision de la scène par l'intermédiaire des cellules photosensibles de la rétine (cônes et bâtonnets). L'influx électrique qui en résulte est envoyé à un relais cérébral où, par l'intermédiaire de synapses, il est transmis à l'aire visuelle du cerveau, située dans le cortex occipital. Le cortex visuel droit traite une partie des informations de l'oeil droit et une partie des informations de l'oeil gauche, et inversement.

FIG. 1.3 – Comment le cerveau recréé la profondeur à partir de deux images stéréoscopiques

sante de l'environnement. D'autre part, beaucoup de casques offre une faible couverture du champ visuel par les écrans, réduisant ainsi le sentiment d'immersion. Enfin, le fait d'être totalement plongé dans l'univers virtuel implique une perte de repères visuels, ce qui peut facilement provoquer un sentiment de «mal du virtuel» comparable au mal de mer. Ce sentiment est d'autant plus important lorsque l'utilisateur «subit» des déplacements involontaires dans son monde virtuel, comme un survol d'un paysage à grande vitesse ou la visite d'un vaisseau sangin.

Actuellement, les casques servent davantage à la réalité augmentée, par le biais d'affichage sur des supports semi transparents ou de reprise vidéo, car ils restent le seul système d'affichage que l'utilisateur peut porter sur lui.

Projection sur grands écrans En prenant l'optique exactement opposée des casques, on peut imaginer placer l'utilisateur dans une pièce spécialisée dont tous les murs seraient d'énormes écrans : c'est le principe de la CAVE^{TM} [CNSD93]. L'utilisateur a alors l'impression d'être en personne dans un endroit virtuel, notamment grâce au fait qu'il continue de voir son propre corps. Notons qu'il est également possible de pénétrer à plusieurs dans la CAVE^{TM}, ce qui n'était pas possible avec un casque. Cette technique se révèle dans la pratique très utile, et a connu de nombreuses variantes, comme les plans de travail virtuels[3], composés d'un ou deux écrans, formant une sorte de bureau virtuel [KF94] ou les murs immersifs[4]. En usant toujours de la projection sur grand écran, d'autres dispositifs ont été imaginés, comme le VisioDome d'eLumens, qui place l'utilisateur devant un écran hémisphériquede couvrir une large partie de son champ de vision avec un seul écran. La figure 1.5 montre quelques uns de ces systèmes.

L'interaction

Outre l'affichage et le sentiment d'immersion, un aspect important de tout environnement virtuel est la possibilité d'interaction entre l'utilisateur et ledit environnement. Comme nous le verrons, la notion d'interaction fait intervenir en plus du matériel des techniques logicielles. Mais puisque l'interaction est une réponse de l'environnement à une initiative de l'utilisateur, il est nécessaire de disposer d'un moyen de communiquer cette initiative à l'environnement par l'intermédiaire d'un périphérique spécialisé. Nous verrons que deux grandes familles de périphériques existent : ceux que l'utilisateur «tient en main», et ceux que l'utilisateur «revêt». A ces périphériques on adjoint la plupart du temps un matériel de suivi de mouvement, permettant de connaitre la position et l'orientation d'un certain nombre de points mobiles dans l'espace, comme la main par exemple.

Le suivi de mouvement permet à la machine de connaître la position et l'orientation de certains points éventuellement mobiles de l'espace. Deux utilisations sont

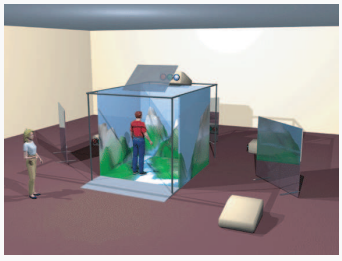
[3] Workbench
[4] Immersive Walls

Le SEOS HMD 120/40

Kaiser Electro-Optics Sim Eye XL100A

Le hi-Res900™3D

FIG. 1.4 – Quelques exemples de HMDs



Une CAVE à 4 écrans (3 cotés et le sol) [Figure de l'Université d'Indiana]

Le VisioDome (Figure E-Lumens)

Le Workbench Consul (Figure Barco)

FIG. 1.5 – Systèmes d'affichage par projection sur grands écrans

principalement faites du suivi de mouvement : le suivi de la tête et le suivi d'un ou plusieurs membres de l'utilisateur. Suivre la position de la tête permet notamment d'adapter l'affichage de l'environnement aux déplacements de l'utilisateur, ce qui est crucial pour une bonne immersion visuelle. Suivre la position d'un ou plusieurs membres va principalement servir à l'interaction, afin de donner l'illusion à l'utilisateur qu'il peut réellement manipuler les objets virtuels comme s'ils étaient réels. D'un point de vue technique, plusieurs grandes familles de technologies sont en concurrence sur cette problématique, chacune avec ses points forts et ses inconvénients. On peut noter le suivi de mouvement par méthode *magnétique* (le *Flock of birds* de la société Ascension [wwwb], ou le *Fastrak* de la société Polhemus [wwwg] par exemple), historiquement le premier, relativement peu onéreux mais très sensible à l'environnement métallique qui provoque des distorsions du champ magnétique sur lequel repose le système. Les technologies à base d'*ultrasons* (de la société Intersense [wwwf] notamment) sont également très répandues, offrant une bonne précision et une plus grande insensibilité à l'environnement. Les méthodes basées sur la reconnaissance optique de formes (de la société ART [wwwa] par exemple) sont également prometteuses, en offrant une très haute précision et une bonne insensibilité à l'environnement. La figure 1.6 donne quelques exemples de périphériques de suivi de mouvement.

Les périphériques tenus en main , la plupart du temps équipés d'un ou plusieurs boutons et éventuellement d'un ou plusieurs joystcks, sont le pendant tridimentionnel de la souris des ordinateurs de bureau. Le mouvement (position et le plus souvent orientation) sont suivis et communiqués à l'application afin de permettre l'interaction. Parmi ce type de périphériques, les plus répandus actuellement sont les *stylus* de *FakeSpace* [wwwc] et les *wand* de *Intersense* [wwwf], mais nombre de variantes existent. La figure 1.7 donne quelques images de ce type de périphériques. Certaines applications spécifiques, comme les simulateurs, reposent sur des périphériques particuliers. Un simulateur de vol disposera par exemple d'un manche à balai et d'une réplique de cockpit, alors qu'un simulateur d'opération chirurgicale pourra disposer d'une réplique de l'outil réellement utilisé par le praticien. Hinckley et ak. [KHK94] montrent une utilisation astucieuse de tels périphériques dans une application d'exploration de données volumiques du cerveau : l'utilisateur tient dans une main une tête de poupée, et dans l'autre une plaque de plexiglas, tous deux étant suivis dans l'espace. La plaque de plexiglas permet de définir aisément un plan de coupe dans les données, alors que la tête de poupée permet de renforcer la corrélation cognitive entre la manipulation et les données manipulées. Ces objets servant à l'interaction portent le nom de *props*.

Les périphériques revêtis par l'utilisateur offrent à l'utilisateur une façon d'interagir dans laquelle le sentiment d'interagir avec une machine est plus réduit. Les gants de données par exemple permettent de connaître un certain nombre d'angles de flexion des doigts, et donc d'en tenir compte durant l'interaction. Selon la per-

formance du gant considéré, on disposera d'une information plus ou moins exacte sur la forme réelle de main. La société 5DT [wwwd] commercialise des gants à faible coût, mais qui ne fournisse qu'un seul angle par doigt, ne permettant pas de distinguer un nombre important de positions. La société Immersion [wwwe] propose une version nettement plus aboutie, capable de renseigner sur 18 ou 22 angles. La société FakeSpace [wwwc] propose quant à elle un gant de donnée uniquement sensible au contact de deux doigts (d'une même main ou non). La figure 1.8 illustre quelques types de gants de données.

D'autres dispositifs plus expérimentaux ont également été proposés, comme celui de Zelenik et al. [ZJFF02]. Les auteurs proposent un dispositif basé sur des boutons contact, que l'utilisateur peut disposer librement sur sa main ou ailleurs. Les boutons sont de taille réduite, ne se tiennent pas en main mais sont solidaire des doigts par exemple. On peut ainsi adapter la disposition physique des boutons aux besoins de l'application. Le dispositif présente l'avantage d'être de taille extremement réduite et se met et s'enlève plus rapidement qu'un gant de données.

1.2.3 L'avenir

De quoi sera faite la réalité virtuelle de demain ? Difficile à savoir. Les principales avancées que l'on attend pour les systèmes d'affichage sont la possibilité de vision stéréoscopique sans port de lunettes –dits *auto stéréoscopiques*, voire des écrans non plus 2D mais directement 3D. Les écrans auto stéréoscopiques commencent à voire le jour, certains ordinateurs sont même équipés de tels dispositifs. Les écrans tridimensionnels, longtemps vus comme la solution ultime à l'affichage d'objets volumiques, commencent également à sortir des cartons. Grâce à eux, l'image n'est plus affichée sur une surface en deux dimension, mais directement en trois dimensions dans un certain volume, par des techniques utilisant des rayons laser ou un écran bidimensionnel en révolution extrêment rapide. Ce type de dispositif permettrait une perception de la profondeur exemplaire qui se passerait de l'affichage d'une image spécifique pour chaque oeil. Ils restent cependant des prototypes, et offrent pour l'instant des performances très limitées.

Concernant le suivi de mouvement, beaucoup de recherches sont faites afin de se passer définitivement des *marqueurs*, ces éléments spéciaux que le système sait reconnaître et permettent le suivi du mouvement. L'argument du prix de ces marqueurs ou de leur encombrement est souvent cité comme motivation à ces travaux. Pour cela, on peut par exemple avoir recours à la reconnaissance de formes à l'aide de multiples prises de vues, pour directement détecter la position de l'extrémité des 10 doigts par exemple. Allard et al. proposent un premier pas vers une telle solution [ABF+04], en faisant de la reconstruction 3D d'un utilisateur à partir de plusieurs caméras. Cependant, une partie importante reste à faire : savoir étiqueter les objets reconstruits, afin de distinguer un bras d'une jambe et de savoir ou se trouvent réellement les doigts où toute autre partie d'intérêt.

Tracking optique
ART [wwwa]

Tracking magnétique
Ascension [wwwb]

Tracking ultrasonique
Intersense [wwwf]

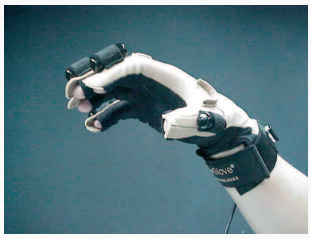
FIG. 1.6 – Périphériques de suivi de mouvement

Wand (InterSense [wwwf])

FlyStick (ART [wwwa])

Stylus (Polhemus [wwwg])

FIG. 1.7 – L'interaction par joystick



Cybertouch [wwwe]
18 ou 22 angles mesurés selon les modèles.

Pinchglove [wwwc]
Sensible au contact entre 2 doigts uniquement

Pop Through buttons [ZjFF02]
Ici disposé sur l'index, mais adaptable sur d'autres doigts et props.

FIG. 1.8 – L'interaction par gant de données

1.3 La réalité logicielle

Comme nous l'avons vu dans les pages précédentes, la réalité virtuelle fait intervenir un nombre important de périphériques matériels spécifiques au domaine. Ces périphériques sont exploités par diverses techniques logicielles, propres aux environnements virtuels. Si la *réalité matérielle* décrit la spécificité matérielle de la réalité virtuelle, la *réalité logicielle* en décrit la spécificité logicielle.

Comme tout système à base d'informatique, un système de réalité virtuelle a besoin de logiciels permettant d'exploiter les différents périphériques. Le logiciel pourra agir comme une couche d'abstraction entre le matériel et un logiciel de plus haut niveau —c'est le cas d'un gestionnaire de périphériques, ou exploiter les capacités du matériel pour les agencer en un paradigme d'interaction. Il est à noter qu'une technique initialement conçue autour d'un périphérique A peut souvent être adaptée à un périphérique B, moyennant parfois une adaptation minimale. Ceci donne une importance d'autant plus grande à la partie logicielle du système, puisque c'est elle qui décidera pour une large part comment tirer profit des périphériques.

Nous allons dans cette partie passer en revue les différentes techniques logicielles dédiées à la réalité virtuelle, selon la taxonomie introduite par Hand [Han97] et formalisée par Bowman [Bow99].

1.3.1 Différentes tâches, différents paradigmes

La réalité virtuelle intègre la notion d'imersion, mais ne serait rien sans la notion d'interaction. L'utilisateur a la possibilité d'agir sur son environnement, tout comme nous le faisons tous les jours dans notre vie quotidienne. Dans la réalité usuelle, nous avons à notre disposition principalement deux types d'interaction (avec l'environnement) :

Se déplacer l'évolution nous a donné la possibilité de nous déplacer dans notre environnement ; cette tâche occupe une importance capitale dans nos activités quotidienne, que ce soit pour faire un mouvement de 50cm vers le côté pour avoir une meilleure vue sur un monument, ou pour rejoindre un lieu à l'autre bout du monde. Associée à cette tâche, on peut noter la nécessité de trouver son chemin, au préalable ou durant le déplacement.

Manipuler des objets à peu près toutes les interactions que nous faisons sur notre environnement passe par la manipulation d'un objet : allumer la lumière, se servir un verre d'eau, écrire, etc. La manipulation est sous cette optique la tâche fondamentale de l'interaction. C'est elle qui permet de réellement changer son environnement.

Ces deux actions, calquées sur notre vie quotidienne, ont donné lieu à la taxonomie suivante, proposée par Hand [Han97] et formalisée par Bowman[Bow99]. Pour eux, toute tâche en réalité virtuelle peut être décomposée en tâches géné-

riques, elles-mêmes réparties en quatre grandes familles : trois proches de la réalité quotidienne, et une dernière plus dépendante du système informatique sous-jacent.

Tâche 1 : Navigation La navigation regroupe l'ensemble des techniques qui permettent à l'utilisateur de se déplacer dans le monde virtuel, ainsi que de trouver son chemin.

Tâche 2 : Désignation La désignation regroupe l'ensemble des techniques permettant à l'utilisateur de mettre en évidence un ou plusieurs objets de la scène. Cette tâche est souvent conjointe à la tâche de *manipulation*.

Tâche 3 : Manipulation La tâche de manipulation regroupe l'ensemble des techniques qui permettent à l'utilisateur d'appliquer un *traitement* à des objets, généralement ceux préalablement désignés. Par *traitement*, on entend aussi bien déplacement ou rotation que traitement plus abstrait comme «activer» un bouton par exemple.

Tâche 4 : Contrôle d'application Le contrôle d'application regroupe toutes les techniques qui permettent à l'utilisateur de manipuler l'application, l'environnement ou les données par voie indirecte. Cette catégorie tient une place particulière, puisqu'elle peut nécessiter elle-même de désigner et manipuler des objets. Dans une certaine mesure, cette tâche se situe à un niveau conceptuel différent des trois autres –l'application manipulée par l'application.

1.3.2 La navigation

Il peut arriver que l'utilisateur ait à se déplacer dans la scène virtuelle. Cela va de déplacer sa tête pour observer un objet sous un autre angle, à se déplacer dans un bâtiment, voire se déplacer de galaxie en galaxie. Les échelles de déplacement peuvent être très différentes, et chaque technique s'adressera à une situation précise.

Dans le cas le plus simple, l'utilisateur souhaite simplement pouvoir se déplacer autour d'un objet pour l'examiner. De tels déplacements sont pris en charge directement par le suivi de mouvement de la tête, il n'est donc pas nécessaire d'y adjoindre une technique spécifique. L'utilisateur n'aura qu'à marcher physiquement, ou pencher sa tête, pour arriver au point de vue souhaité. Cependant, seuls des mouvements de faible envergure peuvent être ainsi pris en charge –une CAVE fait en général 3x3x3m, un workbench autour de 2m de large, etc.. Dans le cas de la visite de lieux plus grands, le suivi de la tête seul ne suffit plus, et d'autres solutions sont à mettre en œuvre.

Puisqu'il est naturel pour un être humain de marcher, cette façon de se mouvoir sera largement utilisée en environnement virtuel. Citons rapidement l'existance de périphériques matériels dédiés à la marche, comme ceux illustés par la figure 1.9, qui permettent à l'utilisateur de marcher physiquement sur place, auxquels nous ne nous intéresseront pas ici.Nous retiendrons plutôt les techniques logicielles proposées, chacune adressant un scénario particulier de déplacement dans le monde virtuel. En effet, la problématique n'est pas la même s'il s'agit de visiter un musée

que s'il s'agit d'aller rapidement inspecter différentes parties d'un bâtiment. On peut globalement diviser les techniques de déplacement en deux grandes familles, selon que l'on connaît ou non sa destination.

Navigation libre

Lorsque l'utilisateur ne connaît pas précisément sa destination, ou que le but de la navigation est précisément d'avoir parcouru le chemin —visite d'un musée par exemple, le déplacement se fait de manière incrémentale, au fur et à mesure du trajet. L'acte de navigation demande alors d'être capable à un instant t de pouvoir déterminer la position de l'utilisateur à l'instant $t+1$. Pour cela, on a une notion de vitesse de déplacement, éventuellement d'accélération, et de direction de déplacement. Il faut bien sur également disposer d'un moyen permettant de démarrer et d'arrêter la navigation.

Usoh et al. ont mené une étude intéressante sur la locomotion en milieu virtuel [UAW+99], dans laquelle ils comparent un déplacement réel (navigation par le tracking), une navigation virtuelle (soucoupe volante) et une navigation totalement libre (vol). Ils montrent ainsi, auprès de 33 volontaires dont un tiers d'habitués des environnements virtuels que les utilisateurs sont le plus habiles lorsqu'ils peuvent utiliser leurs jambes pour se déplacer. A l'opposé, les performances sont les moins bonnes lorsque le déplacement se fait uniquement à l'aide de boutons avant/arrière et droite/gauche.

Direction du mouvement La direction de déplacement peut être obtenue de nombreuses façons. Une façon de faire, directement issue des jeux vidéo, consiste à utiliser un joystick : l'avant et l'arrière pour démarrer le mouvement, et le côté pour tourner. Des boutons peuvent remplacer les quatre directions du joystick. Cependant il est plus interessant d'adapter la technique aux périphériques à 6 degrés de libertés : l'utilisateur peut dans ce cas simplement «montrer» la direction souhaitée, qui sera obtenue par le suivi de mouvement, comme le proposent Butterworth et al. [BDHO92]. On utilise dans ce cas un bouton pour démarrer ou arrêter le mouvement. Cette technique est couramment nommée la métaphore de la *soucoupe volante*[5]. Diverses variantes sont possibles pour montrer la direction de la soucoupe volante. On peut se servir de la direction du regard (*Gaze directed*), de la direction d'un périphérique tenu en main ou de la direction du torse (*Torso directed*) [RH92, Bow99]. Ces deux dernières versions sont préférables, puisqu'elles permettent à l'utilisateur de regarder ailleurs que vers sa destination (pour lire une indication par exemple). Notons cependant que d'après une étude menée par Usoh et al. [UAW+99], les performances ainsi que le sentiment d'immersion sont d'autant meilleures que la métaphore s'approche de la marche réelle, la soucoupe volante donnant les pires performances.

[5] Flying saucer

Razzaque et al. [RSS+02] ont su tirer profit d'un aspect intéressant du déplacement humain. En effet, lorsque l'on cherche à atteindre une destination, la trajectoire est corrigée en permanence de manière imperceptible. Ils utilisent ceci pour permettre à un utilisateur, marchant sur place, de naviguer dans toutes les directions, sans jamais voir les parties sans écran de l'environnement. Pour cela, lorsque l'utilisateur se met en marche dans une direction donnée, la scène est en permanence imperceptiblement tournée de manière à ce que l'utilisateur tende à se diriger vers le mur du fond de leur CAVE –celui qui est le plus loin du mur avant inexistant (voir la figure 1.10 pour une illustration). L'utilisateur bénéficie ainsi d'une plus large amplitude de rotation avant de remarquer qu'il «manque un mur». Cette technique, baptisée *Marche redirigée*[6], donne d'excellents résultats en termes d'immersion et de performance par rapport à la soucoupe volante. La même technique peut être adaptée pour les environnements de type HMD, en faisant en sorte que l'utilisateur se déplace toujours vers le mur physique le plus lointain, évitant ainsi presque toute collision avec les limites réelles de la pièce.

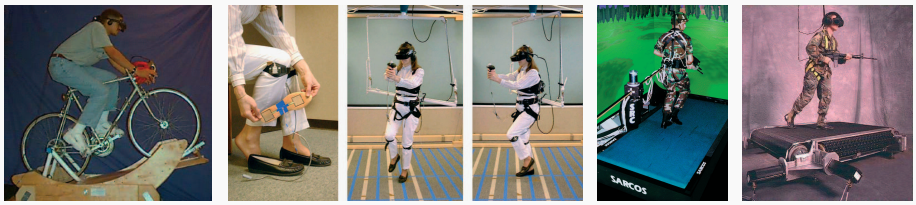
proposent une une technique de navigation qu'ils baptisent «l'attache dans le vide»[7]. En appuyant sur un bouton, la position de l'outil d'interaction est mémorisée. Tant que le bouton'est pas relâché, chaque déplacement/rotation de l'outil provoque un déplacement/rotation identique sur l'ensemble de l'environnement. La manipulation est répétée autant de fois que nécessaire ; cette technique permet donc d'avancer par petits incréments dans l'environnement, ce qui s'avère particulièrement adaptée pour les environnements de taille modérée comme ceux que l'on représente classiquement sur un plan de travail virtuel.

Vitesse de déplacement Outre la direction, le mouvement se caractérise par une vitesse instantanée. Cette vitesse peut être constante tout le long du mouvement, mais il est souvent utile de pouvoir varier celle-ci, pour ralentir près de points intéressants, et accélérer sur les distances plus longues. Song et Norman proposent par exemple un contrôle non linéaire du déplacement [SN92], qui permet dans le cadre d'une visualisation scientifique de se mouvoir précisément à l'intérieur du domaine de simulation, tout en permettant de prendre du recul pour voir le phénomène dans son ensemble.

Une autre approche, proposée par [YAR04], propose de faire marcher l'utilisateur «sur place», et de déduire la vitesse de déplacement à partir de la vitesse de montée/descente des genoux. Pour des cas de déplacement de type marche, on conserve ainsi un fort parallèle avec la façon habituelle de se déplacer, favorisant ainsi l'aisance du déplacement.

Mine et al [MJS97] proposent une technique bimanuelle de déplacement, qu'ils nomment le *Two-Handed Flying*. Les deux bras sont utilisés pour déterminer à la fois direction et vitesse de déplacement : la vitesse est donnée par l'écartement des bras, la direction par le vecteur bissecteur de l'angle formé par les bras.

[6] Redirected Walking
[7] Grab the air



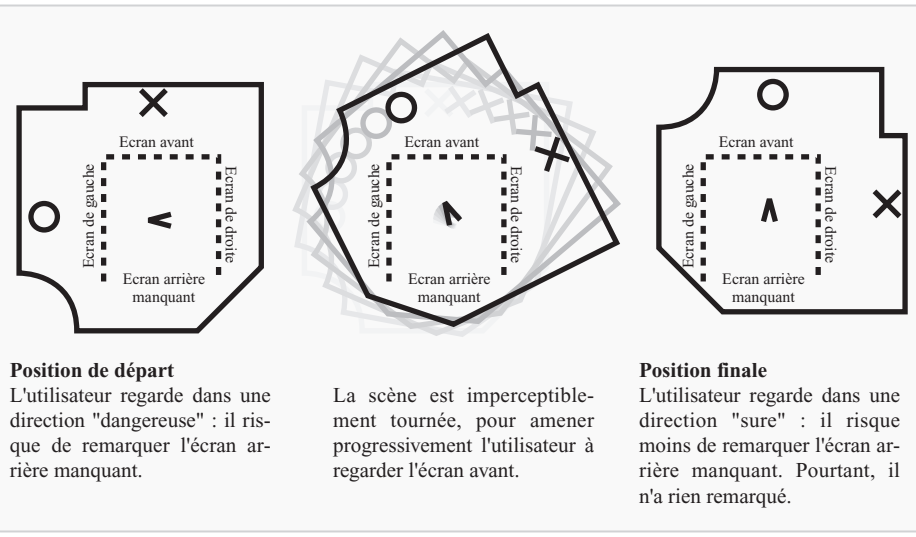
Navigation par cyclisme (Hodgins, Georgia Tech)

Marche sur place suspendue (Templeman's Gaiter system, NRL)

Tapis unidirectionnel (Sarcos)

Tapis multidirectionnel

FIG. 1.9 – Périphériques de marche



Position de départ
L'utilisateur regarde dans une direction "dangereuse" : il risque de remarquer l'écran arrière manquant.

La scène est imperceptiblement tournée, pour amener progressivement l'utilisateur à regarder l'écran avant.

Position finale
L'utilisateur regarde dans une direction "sure" : il risque moins de remarquer l'écran arrière manquant. Pourtant, il n'a rien remarqué.

FIG. 1.10 – Principe de la marche redirigée (d'après [RSS+02])

Meta-navigation Stoackley et al. [SCP95] ont proposé de considérer le problème de la navigation sous une autre perspective : permettre à l'utilisateur de manipuler directement son propre point de vue, en le déplaçant comme il déplacerait un autre objet[8]. Disposant d'un monde en miniature, l'utilisateur y est représenté comme un objet classique, et peut donc être manipulé librement, à ceci près que tout déplacement de l'avatar miniature change le point de vue de l'utilisateur réel comme s'il s'était réellement déplacé.

Navigation assistée

Il arrive également d'avoir à aller d'un point *A* à un point *B*, la destination pouvant être prédéfinie parmi une liste de destinations potentielles. Dans ce cas, la navigation nécessite de pouvoir déterminer la destination, ainsi que le «mode de transport».

Destination Pour déterminer l'endroit où se rendre, plusieurs solutions sont envisageables. Il est possible, lorsque le nombre de destinations est restreint, de les énumérer dans une liste ou un menu, ce qui ne résout le problème qu'à moitié puisqu'il faut alors disposer d'un moyen de désigner un élément dans la liste ou le menu –on repose donc pour cette partie de la tâche sur le contrôle d'application. On peut également citer ici l'utilisation des commandes vocales pour nommer les endroits possibles, quoique l'usage de la voix comme vecteur de commande d'un ordinateur ne soit pas propre à la réalité virtuelle. Une autre technique très utile consiste à utiliser une miniature de la scène[9] [SCP95], et de désigner la destination sur cette miniature. L'utilisation d'une carte 2D est également une possibilité. Elvins [ENSK98] propose une version légèrement différente du monde miniature, en restreignant le contenu de la miniature à certains types d'objets seulement pour éviter la surcharge.

Mode de transport Lorsque le point d'arrivée est connu, il suffit de déplacer l'utilisateur jusqu'à ce point. Ceci peut se faire de manière instantanée, par «téléportation», mais des tests montrent qu'un déplacement instantané désoriente beaucoup l'utilisateur. Il est alors souvent préférable de «faire parcourir» virtuellement le chemin à l'utilisateur. On peut pour cela réutiliser la métaphore de la soucoupe volante, qui cette fois suivra une trajectoire prédéfinie, afin de conserver une continuité spatio-temporelle et ainsi minimiser la désorientation de l'utilisateur. Selon le type de scène, on pourra adapter cette technique en contraignant la trajectoire : dans un bâtiment, on souhaitera par exemple que la trajectoire soit «plausible», c'est à dire qu'elle ne traverse pas les murs et suive les couloirs du bâtiment. La vitesse de parcours de la trajectoire peut elle aussi être calculée de diverses manières : constante quelle que soit la longueur du trajet (vitesse=const,

[8] Serait-ce une application littérale de l'expression «jeter un œil» ?

[9] WIM : World In Miniature

durée=f(trajet)), constante mais adaptée à chaque trajet (vitesse=f(trajet), durée=const), non constante (vitesse=f(position), duree=$f(trajet)$)[Min95b].

1.3.3 La désignation

L'acte de désignation est à la base d'un système interactif. En effet, si l'utilisateur doit être en mesure d'agir sur ce qui l'entoure, il doit disposer d'un moyen lui permettant d'exprimer ce sur quoi il veut agir. Cette action est souvent suivie d'une action de manipulation (voir section suivante), dans laquelle il va effectivement modifier l'environnement.

Afin de désigner un ou plusieurs objet(s) dans la scène, on peut avoir recours à diverses techniques, selon que les objets composant la scène soient grands, petits, lointains, proches, qu'il y ait beaucoup d'objets, peu, etc..

Nous allons ici détailler les différentes techniques que l'on peut rencontrer dans la littérature, en les regroupant d'un point de vue utilisateur.

Désignation par préhension

Définition La désignation par préhension consiste à utiliser la main de l'utilisateur, ou une représentation virtuelle de celle-ci, pour désigner l'objet.

On peut choisir de désigner l'objet d'intérêt de manière analogue à la réalité quotidienne : en le «prenant». Pour cela, il est commun d'effectuer un geste de «poing» à l'intérieur de l'objet ou de laisser la main un certain temps dans l'objet [SZP89]. Cette technique, quoique simple et intuitive, pose le problème de la désignation des objets situés à «plus d'un bras» de l'utilisateur, pour lesquels il faut au préalable se déplacer.

Poupyrev et al. ont proposé pour atténuer ce problème une technique connue sous le nom de *GoGo*[10] [PBWI96]. Leur métaphore introduit la notion de main virtuelle, dissociée de la main réelle. La position de cette main virtuelle est fonction de la distance entre la main réelle et l'épaule (réelle) par une fonction en partie non linéaire, ce qui permet d'atteindre des objets situés plus loin. La figure 1.11 illustre cette métaphore. L'utilisateur dispose ainsi d'un bras virtuel plus long que son bras réel, lui permettant d'atteindre des objets plus éloignés, mais reste limité tout de même.

Bowman propose plusieurs variantes de la métaphore du Go-Go dans sa thèse [Bow99], notamment le *Fast GoGo*, permettant d'atteindre des objets encore plus éloignés, au détriment de la précision et le *Stretched GoGo*. Cette métaphore propose, toujours sur la base de la distance entre main réelle et épaule, de diviser l'intervalle en trois zones : proche, neutre et éloignée. Lorsque la main se situe en zone proche, la main virtuelle va s'approcher, à vitesse constante. Lorsque la main se situe en zone éloignée, la main virtuelle va s'éloigner, à vitesse constante. Enfin,

[10] Du nom de l'anti-héros *L'Inspecteur Gadget*, un agent cybernétique, pouvant entre autre allonger son bras en prononçant les mots «GoGo Gadget au Bras»

lorsque la main se situe en zone neutre, la main virtuelle n'avancera ni ne reculera. Cette technique permet d'atteindre des objets aussi éloignés soient-ils, mais s'avère dans la pratique très difficile à manipuler.

La métaphore *Head Crusher*[11][PFC+97] propose une autre vision de la sélection par préhension. L'utilisateur va, à l'aide du pouce et de l'index, effectuer un mouvement de «pince» pour «prendre» l'objet qu'il désire, comme s'il agissait sur l'image perçue plutôt que sur l'objet réel. D'autres techniques, basée sur le même concept d'interagir sur l'image perçue de l'objet sont également proposées par les auteurs. La métaphore des *Framing Hands*[12] permet de définir, en plaçant les mains de sorte à former un cadre, La figure 1.12 illustre ce principe. Cette famille de techniques porte le nom de *Image Plane Interacion Techniques*, car l'utilisateur manipule les objets comme il les perçoit en 2D et non directement en 3D.

Désignation indirecte

Définition La désignation indirecte consiste à désigner l'objet non pas en le montrant directement, mais en agissant sur une représentation secondaire de lui-même.

Il est possible d'énumérer tous les objets dans une liste ou un menu, et d'opérer la sélection dans ce menu. A nouveau, comme pour le choix d'une destination de déplacement, le problème n'est qu'à moitié résolu. Cependant, une telle technique permet d'éviter toute ambiguïté quant à la sélection sous réserve d'être capable de nommer chaque objet de façon non équivoque.

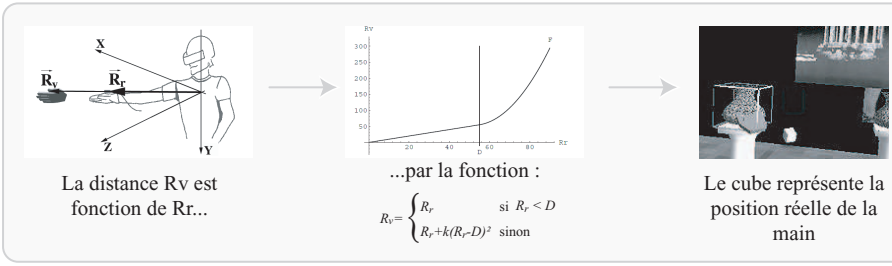
Une variation intéressante de l'énumération nominative est proposée par Stoackley et al. : le monde en miniature[13][SCP95]. L'utilisateur dispose d'une maquette de la scène, soit en main soit à proximité de lui, sur laquelle il pourra désigner l'objet qui l'intéresse. Cela rend possible des sélections d'objets hors de portée, et se passe de donner un nom à chaque élément. La figure 1.13 illustre l'utilisation d'un WIM.

Désignation par pointage

Définition La désignation par pointage consiste à montrer le ou les objets d'intérêt à distance.

La désignation par pointage est une technique très répandue, notamment parce qu'elle permet de désigner des objets éloignés sans avoir à se déplacer au préalable. La technique du lancer de rayon[14] fut la première de cette famille[Bol80], reprise par [Min95b, Min96]. L'auteur propose de lancer un rayon virtuel au travers de la scène, dans la direction indiquée par une sorte de stylo tenu par l'utilisateur. Ce

[11] Écraseur de tête
[12] Mains encadrantes
[13] WIM, World In Miniature
[14] Raycasting



Le vecteur R_v, définissant la position de la main virtuelle, est défini par rapport au vecteur R_r, mesuré par deux trackers (l'un sur le torse, l'autre au poignet), mais sa norme est modulée par une fonction non linéaire.

FIG. 1.11 – La métaphore du *Go-Go* (images de [PBWI96]).

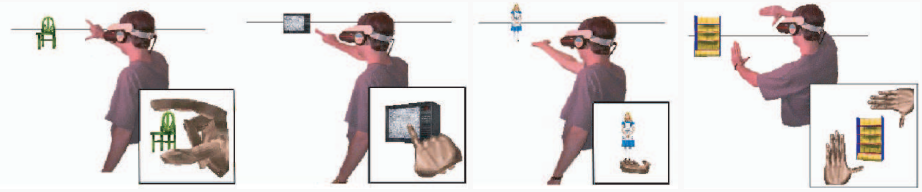


FIG. 1.12 – La métaphore du *Head Crusher* (images de [PFC+97]).

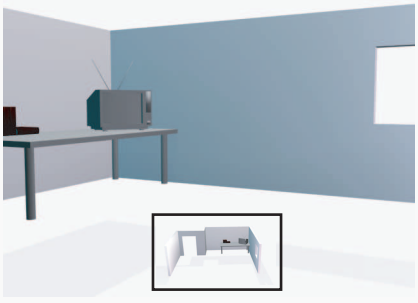


FIG. 1.13 – Le *monde en miniature* (d'après [SCP95]).

rayon va intersecter un ou plusieurs objets au cours de son trajet dans la scène ; le premier objet rencontré sera retenu comme sélectionné. Pour aider à la sélection, il est courant de matérialiser la direction du rayon par une ligne, comme un rayon laser (d'ou le nom classique de cette technique, le *laser virtuel*). La figure 1.14 illustre la métaphore originale. L'utilisateur a effectivement la sensation de tenir en main un pointeur laser, dont le faisceau sert à sélectionner un objet. Cette métaphore a l'avantage d'être très aisée à prendre en main en raison du parallèle très fort avec la manipulation d'une lampe de poche, mais souffre d'une relative imprécision –amplification des tremblements de l'utilisateur par la distance, notamment pour les objets de petite taille ou éloignés. Plusieurs métaphores dérivées ont été proposées, pour circonvenir à ces inconvénients. Liang et al. proposent d'utiliser un cône plutôt qu'un rayon [LG94], en partant du principe que si les objets éloignés deviennent de plus en plus petits avec la distance, alors l'outil de sélection devrait lui devenir de plus en plus grand. Forsberg et al. proposent de permettre à l'utilisateur de modifier comme il le souhaite l'angle d'ouverture du cône [FHZ96], en une métaphore qu'ils nomment le *laser à ouverture*[15]. Cela permet d'adapter la précision de la sélection à la situation : pour les objets éloignés, on utilisera un cône plus large que pour les objets proches. Une seconde variante est également présentée, permettant l'utilisation d'une pyramide conique plutôt que circulaire comme volume de sélection.

Zhai et al. [ZBM94] ajoutent à l'extrémité du rayon un volume de sélection, qui permet de maximiser la loi de Fitt : Le temps de sélection est d'autant plus petit que la surface (ou le volume) de sélection est grande. Le volume lui-même est affiché en transparence afin d'éviter les occlusions. Les auteurs concluent cependant que la technique donne de meilleures performances lorsque la scène est affichée avec un haut réalisme, qui permet de mieux apprécier la profondeur.

La métaphore du rayon laser a connu d'autres variantes, dont beaucoup portent sur la façon dont le rayon est obtenu –point de départ et direction. Pierce et al. [PFC+97] proposent le laser *dirigé par le regard*, dont l'origine se situe sur la tête, et la direction passe par l'index de l'utilisateur. Subjectivement, l'utilisateur à la sensation de «montrer» ce qu'il veut sélectionner. Les auteurs baptisent la technique le *sticky finger*. Cependant, la métaphore requiert une excellente calibration des périphériques. Le figure 1.12 et 1.15 illustrent le principe de ces métaphores. Olwal et al. [OF03] proposent eux un rayon que l'utilisateur peut courber, afin d'atteindre des objets partiellement cachés par d'autres.

Une autre approche basée sur un principe similaire a été proposée par Steed et Parker [SP04]. Leur idée est la suivante : plutôt que de désigner l'objet que l'on souhaite sélectionner, l'utilisateur va petit à petit éliminer tous les objets qu'il ne veut pas sélectionner. Pour ce faire, ils proposent d'inverser le comportement

[15] Aperture laser

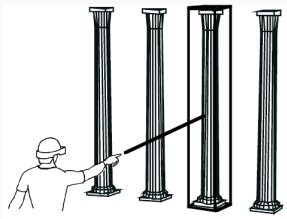


Image de [Min95]

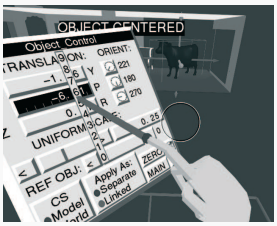
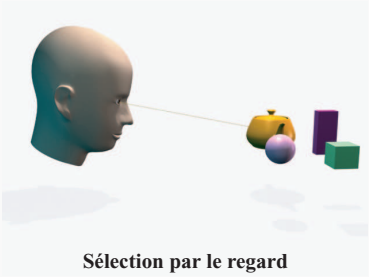
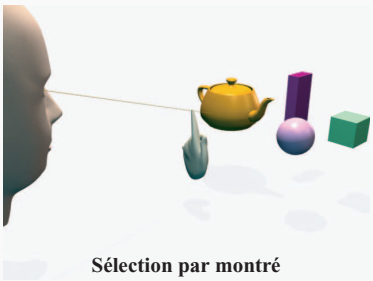


Image de [Min96]

FIG. 1.14 – La métaphore du *Laser*.



Sélection par le regard
Rayon partant de l'oeil en direction du regard.



Sélection par montré
Rayon partant de l'oeil en direction de l'extrémité de l'index.

FIG. 1.15 – Variantes du laser.

du rayon laser : l'utilisateur tient en main une *lampe de poche virtuelle*, dont le faisceau va «éclairer» certains objets, qui seront marqués comme potentiellement sélectionnés. Pour réduire le nombre d'objets sélectionnés, l'utilisateur va faire progressivement sortir du faisceau les objets qu'il ne désire pas conserver, en effectuant de petits mouvements avec la lampe torche virtuelle. Les objets sélectionnés au final seront ceux qui ne sont jamais sortis du faisceau de la lampe au cours de la manipulation. Des tests auprès d'utilisateurs montrent que cette technique est fonctionnelle et évite un grand nombre d'erreurs de sélection, notamment dans le cas où la personne qui manipule n'est pas la personne qui dispose du suivi du regard[16].

1.3.4 La manipulation

L'acte de manipulation va en général de paire avec l'acte de désignation, opérant sur celle-ci. Nous allons voir qu'il existe différents types de manipulations, et différentes manières d'effectuer ces manipulations, qui sont souvent très liées à la technique de sélection employée. Le tableau 1.1 donne quelques éléments de comparaison des différentes techniques de désignation et de manipulation.

Dans un système de réalité virtuelle, on imagine effectuer des manipulations aussi diverses que variées, sur des objets tout aussi hétéroclites. Cependant, en y regardant de plus près, on s'aperçoit que ces manipulations, aussi complexes soient-elles, peuvent la plupart du temps se ramener à une composition de manipulations simples, éventuellement contraintes. Par manipulation simple, nous entendons ici les manipulations de base consistant à déplacer un objet : bouger et/ou tourner. Ces actions peuvent éventuellement être contraintes, de manière plus ou moins sophistiquées (bouger le long d'une droite, d'un plan, d'une courbe, tourner selon un axe uniquement, etc.)

Manipulations locales

Définition Une manipulation est dite locale lorsqu'elle a les mêmes comportements qu'une manipulation effectuée a portée de main. Ces manipulations qualifiées de *centrées objet*[17].

Lorsque nous saisissons un objet dans la réalité quotidienne, nous effectuons une manipulation locale : l'objet manipulé est au même endroit que le manipulateur. Ceci permet d'effectuer des mouvements précis, puisqu'il y a une transmission 1 : 1 entre la main et l'objet, en particulier pour les rotations, qui se font autour du poignet. La rotation est ainsi découplée –ou presque– de la translation.

[16] Ce cas de figure se produit notamment pendant une démonstration à un ou plusieurs visiteurs, durant laquelle la personne qui manipule cède le suivi de mouvement au visiteur, mais effectue tout de même quelques manipulations.

[17] Object centered

Parmi les métaphores proposant une manipulation centrée sur l'utilisateur, on peut citer la manipulation directe, qui au final ne fait intervenir aucun intermédiaire entre le périphérique d'interaction et l'objet manipulé. D'autres techniques, comme le Go-Go [PBWI96], proposent également une manipulation locale, mais au bout d'un bras à rallonge. La technique du monde en miniature permet également une manipulation locale, en opérant sur la miniature plutôt que sur l'objet complet, toute modification opérée sur la miniature étant transmise à la scène.

Bowman et Larry ont proposé une technique intéressante, alliant l'efficacité de la sélection par pointage et la précision de la manipulation locale : la technique HOMER[BH97], basée sur les travaux de [WG95] et [Min95a]. La sélection est faite par pointage (raycasting ou dérivé), mais la manipulation est faite à la manière du Go-Go, donc localement à l'objet. Éventuellement, la métaphore *Homer Fishing Rod*[Sch03] permet à l'utilisateur de rapprocher ou d'éloigner l'objet de sa main, à la manière d'une canne à pêche, de manière à choisir au cas par cas s'il souhaite manipuler l'objet à distance ou près de sa main, en se servant de la main non dominante pour contrôler la distance. La figure 1.16 illustre ce principe.

Pierce et al. proposent une sorte de variation du *monde miniature* pour la manipulation, qu'ils nomment les poupées vaudou[18][PSP99] (figure 1.17). Cette technique propose, comme le *WIM*, de manipuler des clones miniature, à la différence que les miniatures sont créées à la demande. Pour cela, l'utilisateur désigne l'objet qu'il souhaite manipuler par la technique *head crusher* (voir figure 1.12). Une miniature de l'objet et de son environnement proche est alors créée dans la main non dominante –qui sert donc de support à la manipulation, la main dominante servant à déplacer et tourner la miniature. La technique permet notamment la manipulation d'objets de taille variées –pourvu qu'il soit possible de les désigner, et ce à proximité de soi.

Kitamura et al. [KHMK99] proposent une méthode de manipulation originale basée sur l'utilisation de baguettes chinoises, permettant de saisir, déplacer et tourner les objets (figure 1.18). Selon le cas, l'une des baguettes peut servir d'axe de rotation tandis que l'autre indique l'amplitude de la rotation, ou alors les deux baguettes définissent un angle et un centre implicite.

Enfin, Ware et Rose utilisent des objets réels, de formes similaires aux objets virtuels, sur lesquels ils effectuent les rotations et déplacements [WR99]. L'utilisateur dispose ainsi d'un retour tactile cohérent par rapport à ce qu'il fait (tourner un objet) et ce qu'il ressent (l'objet dans sa main). Leurs tests montrent que la précision et la performance sont meilleurs avec ces *props* que sans.

Manipulations distantes

Définition Une manipulation est dite distante si elle est appliquée à partir d'un centre autre que le centre de l'objet manipulé. Ce type de manipulation est généra-

[18] Voodoo dolls

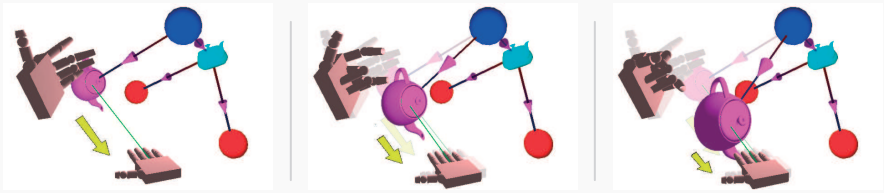
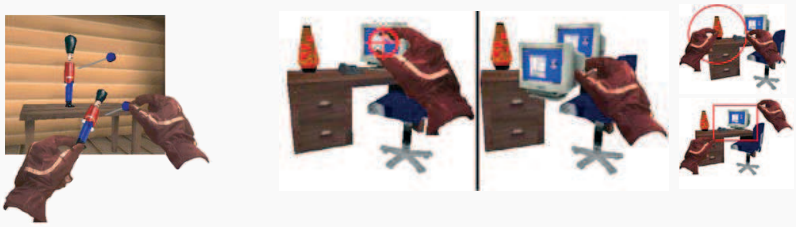


FIG. 1.16 – La métaphore *Homer Fishing Rod* (images de [Sch03])



Poupée vaudou : le soldat tenu par la main gauche sert de référence au placement de l'aiguille manipulée par la main droite.

Création du contexte de référence pour la manipulation d'une poupée vaudou. Au centre, par cible. En haut à droite, par encerclement, en bas à droite par encadrement.

FIG. 1.17 – Les *poupées vaudou* (images de [PSP99])

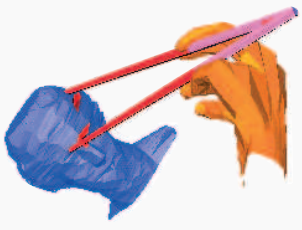


FIG. 1.18 – Manipulation par *baguettes* (image de [KHMK99])

lement appelé «centré sur l'utilisateur»[19].

Dans une scène virtuelle, il est souvent fastidieux d'avoir à se déplacer avant de pouvoir désigner puis manipuler un objet. Aussi, on peut choisir de manipuler un objet directement à distance –sorte de *télékinésie*, sans avoir à quitter sa position courante. La façon de faire la plus courante est de procurer à l'utilisateur un rayon, sur lequelle il embroche l'objet pour le manipuler. La figure 1.19 illustre les effets d'une manipulation locale et distante sur un objet.

Les sélections distantes sont le plus souvent reliées aux métaphores de sélection par pointage que nous avons vus dans la partie *Désignation*. Puisque l'objet est comme embroché sur ce rayon imaginaire, tout mouvement de celui-ci emportera l'objet avec lui. De par le facteur de distance entre la main de l'utilisateur et l'objet, les déplacements sont très rapides (et éventuellement imprécis). Les rotations se feront quant à elles autour de l'utilisateur, puisque le rayon part de lui. Pour contempler l'arrière d'un objet, plusieurs manipulations sont nécessaires (même si l'utilisateur était capable de tourner son poignet de 180°, l'objet se retrouverait derrière lui). De même, l'utilisateur aura tendance à tourner sa main pour faire une translation, en profitant du facteur de distance , ce qui en plus de déplacer l'objet lui appliquera également une légère rotation. On peut simplement citer la technique HOMER, déjà évoquée plus haut, qui propose une manipulation centrée objet (associée à une sélection de type pointage).

Manipulation contrainte

Définition Une manipulation est dite contrainte lorsque seule une partie des mouvements de l'utilisateurs sont transmis à l'objet manipulé.

Nous avons évoqué jusqu'ici les deux manipulations de base : translation et rotation. Ces manipulations étaient jusqu'ici libres de toutes contraintes, c'est à dire que l'utilisateur pouvait déplacer ou tourner les objets sélectionnés comme il le souhaitait. Cependant, il existe bon nombre de situations où certains déplacement sont interdits : coulisser un anneau sur un axe, tourner une roue, etc. On a dans ce cas recours à des contraintes[20], qui vont limiter les possibilités de mouvement. Toutes les techniques citées précédemment sont adaptables : il suffit de «couper» les mouvements indésirables pour s'assurer du respect des contraintes. On peut parfois également préférer une manipulation indirecte, par le biais du contrôle d'application (cf. 1.3.5), par exemple en déplaçant un curseur au lieu de déplacer directement l'objet, auquel cas le respect des contraintes sera plus facile à garantir.

Kiyokawa et al. [KTY96] proposent un système de contraintes discrètes, qui consiste à n'autoriser les déplacements et rotations que selon certaines directions

[19] user centered

[20] Les contraintes évoquées ici restent simples, comme le déplacement le long d'un axe ou d'une courbe, ou la rotation selon un ou deux axe(s) seulement

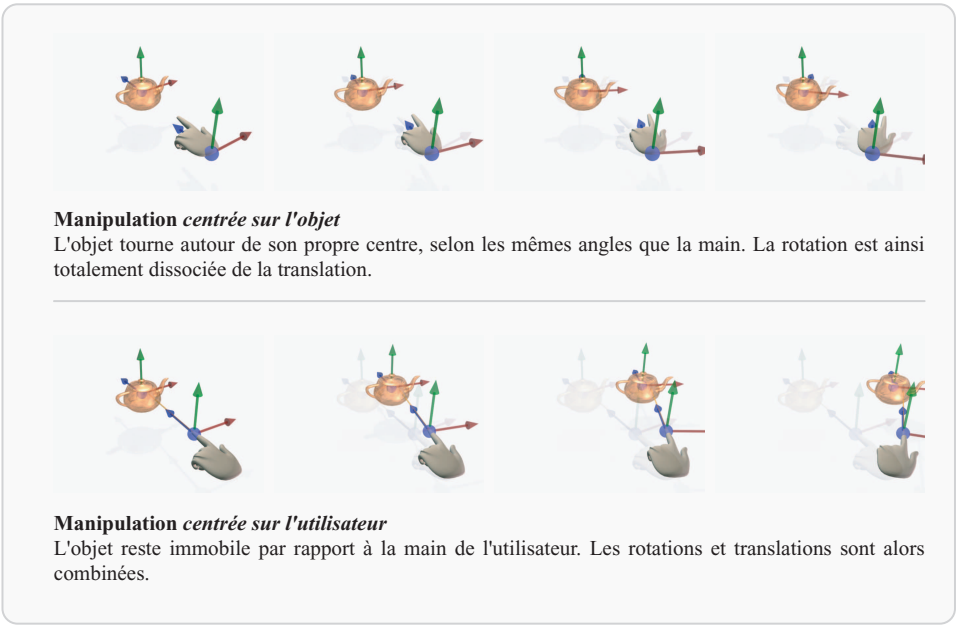


FIG. 1.19 – Manipulation centrée objet et utilisateur

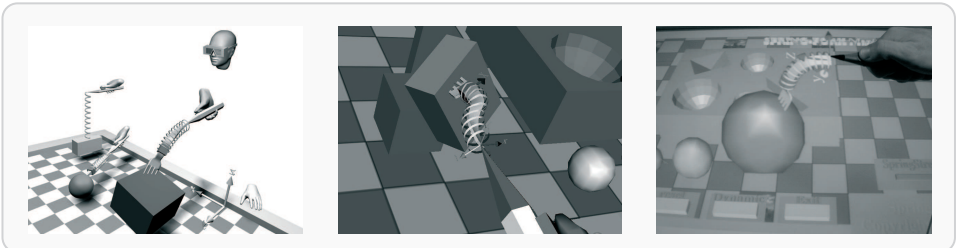


FIG. 1.20 – Manipulation par *ressorts virtuels* (images de [KP01])

ou certains axes, et d'une certaine amplitude maximale. Leur idée est de se rapprocher d'avantage d'une manipulation réelle, constamment soumise à des limites (limites physiologiques des articulations par exemple). Miller et al. reprennent et généralisent quelque peu ce type de contraintes, en proposant une liste d'objets type ainsi que les contraintes associées qu'ils jugent souhaitables [MZ99]. Ils proposent par exemple un outil de déplacement parallèle, contraignant le déplacement dans un plan. Kallmann et Thalmann [MK99] utilisent également des contraintes, afin de limiter les actions possibles sur certains éléments (par exemple, une porte ne peut que s'ouvrir et se fermer autour de ses gonds). Ils proposent de plus d'articuler tout leur système autour de ces contraintes : puisqu'une porte ne peut avoir que deux états (ouverte et fermée), l'interaction se limite à passer la porte d'un état à l'autre.

Gösele et Stuerzlinger utilisent un système de contraintes sémantiques [GS99] pour définir les déplacements autorisés d'objets, afin de garantir un placement cohérent d'objets dans une scène. Le même concept est utilisé par Smith [SSS99].

Koutek et Post proposent une autre méthode innovante de manipulation [KP01] (figure 1.20). L'objet manipulé est attaché à une extrémité d'un ressort virtuel, dont l'autre extrémité est tenue en main par l'utilisateur. En temps normal, le ressort transmet les mouvements à l'objet (avec un petit temps de retard). Mais au besoin, pour signifier par exemple un mouvement impossible à réaliser, le ressort peut se raidir ou se détendre, donnant ainsi une sensation de difficulté à l'utilisateur. Leur métaphore est utilisée dans une application de simulation moléculaire, dans laquelle l'utilisateur peut déplacer des atomes, mais en respectant certaines contraintes physiques. Lorsqu'un déplacement serait contraire aux lois physiques, le ressort permet de le communiquer à l'utilisateur en se raidissant ou se détendant adéquatement.

1.3.5 Le contrôle

Comme toute application informatique, une application de réalité virtuelle a besoin d'être pilotée par l'utilisateur. Le contrôle d'application regroupe l'ensemble des techniques proposant une interface de communication adaptée entre l'utilisateur et le programme qu'il manipule. Une comparaison réductrice peut être faite entre l'interface utilisateur d'un programme et le contrôle d'application. Les deux domaines sont similaires, mais le contrôle d'application englobe davantage de techniques. Si l'on tentait de donner une définition du contrôle d'application, on pourrait dire que ce sont toutes les techniques qui permettent à l'utilisateur de modifier des états internes de l'application.

Le contrôle d'application est une tâche spéciale : elle permet à l'aide de l'application de contrôler cette même application. A ce titre, elle repose sur les tâches de sélection et manipulation, mais en introduisant souvent des manipulations spécifiques, souvent très contraintes en degrés de liberté, afin de coller au plus près à la tâche à accomplir. Parmi les tâches courantes du contrôle d'application, on peut citer l'envoi d'une commande à l'application, le changement d'une valeur, le choix

d'un élément dans une liste, etc.

D'une manière générale, le contrôle d'application en réalité virtuelle peut être abordé sous deux optiques : par l'adaptation à la 3D des techniques utilisées sur les ordinateurs de bureau, ou par la création de techniques inédites exploitant mieux les possibilités nouvelles offertes par un environnement virtuel (degrés de liberté, immersion, etc.). Nous allons dans cette partie détailler les principales métaphores de contrôle d'application trouvées dans la littérature de réalité virtuelle, avec un intérêt tout particulier pour les métaphores de menus.

Les interfaces 2D

Définition On nomme interface 2D les interfaces graphiques des ordinateurs de bureau.

En tant qu'utilisateurs d'ordinateurs, nous disposons tous d'une certaine habitude des interfaces graphiques «modernes», basées sur le paradigme du *WIMP*[21]. L'origine de telles interfaces remonte au début des années 1980 [SIKH82], mais n'ont été rendues populaires que quelques années plus tard, notamment par le fabriquant MacIntosh, et sont aujourd'hui reprises par la majorité des interfaces graphiques. Les interfaces WIMP sont devenues un standard qui s'est imposé de lui-même. L'idée de ces interfaces repose sur le concept de *Widget*[22], associant une représentation graphique et un comportement [CSH+92a], et d'un curseur de désignation. L'ensemble de l'interface est obtenue en composant avec ces widgets.

Selon la tâche à accomplir, l'un ou l'autre widget sera utilisé, soit parmi les classiques que l'on retrouve partout, soit en en inventant de nouveaux. Voici, à titre d'exemple, quelques tâches et les widgets couramment utilisés.

Envoyer une commande à l'application se fait par l'utilisation d'un menu, d'un bouton de commande ou d'une barre d'outil.

Entrer une ligne de texte se fait avec un champ texte (et un clavier).

Entrer une valeur numérique se fait avec un champ texte (éventuellement adjoint de boutons + et -, et un clavier).

Sélectionner une valeur discrète dans un intervalle se fait avec une potentiomètre.

Activer/Désactiver une option se fait avec une case à cocher.

Choisir une option dans un jeu exclusif se fait à l'aide de boutons radio.

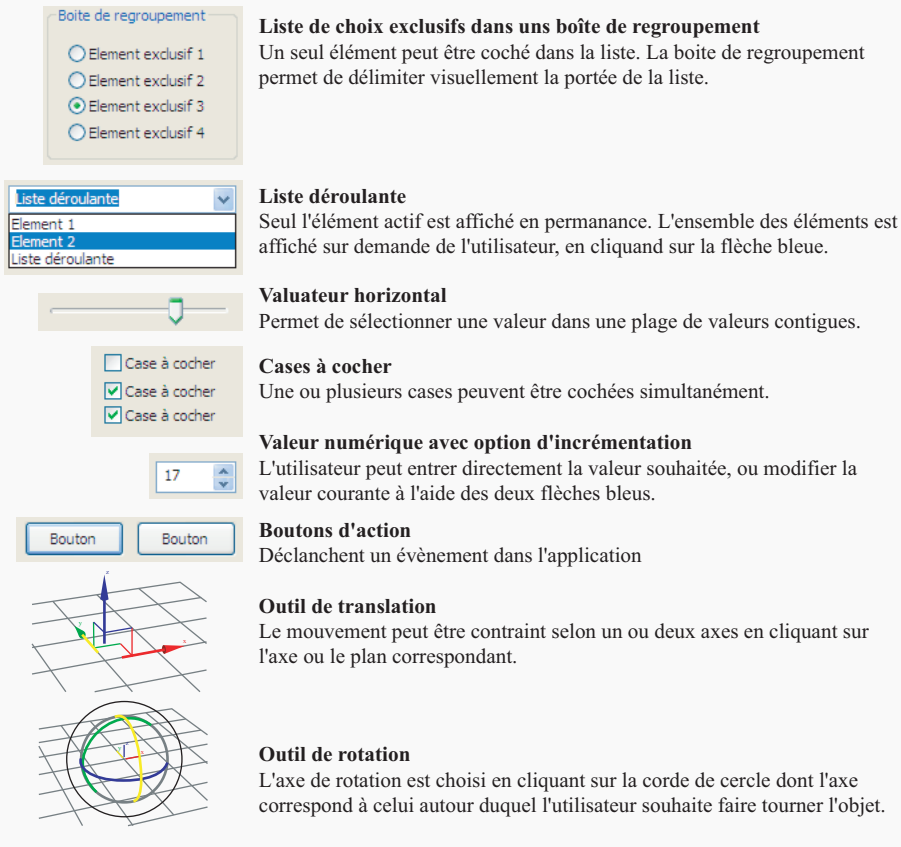
Déplacer un objet 3D se fait en utilisant un widget de translation (figure 1.21).

Tourner un objet 3D se fait en utilisant un widget de rotation (figure 1.21).

[21] WIMP : Windows, Icons, Menus and Pointer, ou Fenêtres, Icônes, Menus et Curseur
[22] Widget : Window Object, ou Composant Graphique

Métaphore	Désignation	Manipulation	Limitations
Liste énumérée	Indirecte	-	Nommage des objets Taille de la liste
World in Miniature	Préhension	Centrée objet	Petits objets Scènes denses
Extension du bras (Go-Go, etc.)	Préhension	Centrée objet	Objets éloignés
HOMER	Désignation	Centrée objet	Scènes denses Objets petits ou très éloignés
Image-Plane (Head Crusher, etc.)	Préhension	-	-
Ray-Casting (Laser, etc.)	Désignation	Centrée utilisateur	Scènes denses Objets petits ou très éloignés

TAB. 1.1 – Comparaison des principales familles de métaphores de sélection/manipulation



Liste de choix exclusifs dans uns boîte de regroupement
Un seul élément peut être coché dans la liste. La boite de regroupement permet de délimiter visuellement la portée de la liste.

Liste déroulante
Seul l'élément actif est affiché en permanance. L'ensemble des éléments est affiché sur demande de l'utilisateur, en cliquand sur la flèche bleue.

Valuateur horizontal
Permet de sélectionner une valeur dans une plage de valeurs contigues.

Cases à cocher
Une ou plusieurs cases peuvent être cochées simultanément.

Valeur numérique avec option d'incrémentation
L'utilisateur peut entrer directement la valeur souhaitée, ou modifier la valeur courante à l'aide des deux flèches bleus.

Boutons d'action
Déclanchent un évènement dans l'application

Outil de translation
Le mouvement peut être contraint selon un ou deux axes en cliquant sur l'axe ou le plan correspondant.

Outil de rotation
L'axe de rotation est choisi en cliquant sur la corde de cercle dont l'axe correspond à celui autour duquel l'utilisateur souhaite faire tourner l'objet.

FIG. 1.21 – Quelques widgets d'interface 2D

La 2D$^{\frac{1}{2}}$, ou l'extension de la $2D$ à la $3D$

Définition On nomme interface 2D$^{\frac{1}{2}}$ toute interface plane issue des ordinateurs de bureau que l'on injecte dans l'environnement 3D.

Afin d'amener le contrôle d'application en réalité virtuelle, on peut choisir d'étendre ou d'adapter certains widgets 2D à la 3D. Brookshire et al. lancent la tendance [CSH+92b], en explicitant une définition du widget comme étant une combinaison de géométrie et de comportement. Partant de là, ils construisent une bibliothèque de composants : sphère contrôlant les rotations, sélecteur de couleurs cubique, poignées, etc.

Jacoby et Ellis proposent, en 1992, un menu déroulant $2D^{1/2}$[JE92], nommé *menu 2D converti*. Ces menus sont librement positionables et orientables dans l'espace. L'utilisateur peut, grâce à une métaphore de laser virtuel, pointer du doigt l'élément qu'il désire activer (figure 1.22). Darken [Dar94] ont amélioré le principe en rendant les menus translucides et en adjoignant un retour haptique à la manipulation du menu. De la même manière, il est possible d'adapter le concept de fenêtre et de widget, ce qui a comme principal avantage de permettre à l'utilisateur d'organiser son environnement en trois dimensions plutôt qu'en deux. Par exemple, pour contraindre une translation selon un axe, on peut recourir au même composant qu'en 2D (figure 1.21, «outil de translation»), à la différence que la sélection se fera par rayon laser plutôt qu'à la souris [Sch03], de même que la manipulation. On peut même recourir aux deux mains, l'une pour spécifier l'axe de contrainte, l'autre pour donner l'amplitude du mouvement.

En injectant ainsi la 2D dans le monde 3D, on dispose d'une bibliothèque importante de composants, adaptés à des situations très variées. Cependant, une telle approche pose le problème de l'adéquation de la métaphore à l'environnement. Les widgets que l'on injecte dans le monde virtuel ont été conçus pour un environnement bien spécifique, celui ou un utilisateur dispose d'un dispositif de pointage précis (sa main, reposant sur une table, et tenant la souris). Or, l'utilisateur d'un environnement virtuel est le plus souvent mobile, et sa main ne repose sur aucune surface. La précision de sa désignation est donc nettement moins précise que celle de l'utilisateur d'un ordinateur de bureau. Afin de pallier à ce problème, on peut agrandir la taille des composants, au prix de se retrouver confronté à un manque de place dans l'espace d'affichage. On peut également de seconder les composants graphiques par des objets réels, comme des plaques de plexiglas servant d'écrans virtuels (ce sont là encore des *props*). L'utilisation de *props* dans ce contexte permet de procurer un retour tactile à l'utilisateur [SES99] (l'utilisateur appuie bel et bien sur quelque chose lorsqu'il appui sur un bouton). Outre le retour tactile, le *props* permet de rapprocher l'interface de l'utilisateur, donc d'augmenter la précision et de diminuer la taille des widgets. Enfin, L'utilisation de périphériques portables, comme les assistants personnels, est également une solution pour y afficher et manipuler l'interface de contrôle [WDC99].

La 3D

Définition On appelle interface 3D un paradigme conçu explicitement pour un environnement manipulé en trois dimensions.

A l'opposé de la vision adoptée dans le paragraphe précédent, on peut proposer des techniques dédiées à la réalité virtuelle, qui n'ont qu'un pendant très lointain à leur équivalent 2D , voir même aucun. De telles techniques visent à tirer d'avantage profit des périphériques et des degrés de libertés supplémentaires offerts par la réalité virtuelle, de même que de tirer profit de sens tels que la proprioception[23]. Les travaux de Mine et al. [MJS97] illustrent parfaitement l'usage de la proprioception pour effectuer certaines tâches. Par exemple, pour supprimer un objet, ils proposent de le «jeter derrière soi». Symétriquement, l'action peut être annulée en allant simplement le «rechercher» au dessus de son épaule (figure 1.23). Leurs techniques sont particulièrement intéressantes, puisqu'elles se passent d'interaction avec le monde virtuel et tirent totalement profit des connaissances que l'on a de son propre corps.

Le contrôle d'application en réalité virtuelle est une discipline jeune, aux avancées relativement peu nombreuses par rapport à son équivalent bidimensionnel. Seules quelques métaphores de contrôle spécifiquement dédiées à la réalité virtuelle ont été proposées, principalement des métaphores de menu. Rappelons qu'un menu a pour but de permettre à l'utilisateur de choisir un élément –généralement une commande– parmi un ensemble de commandes généralement sémantiquement liées. Nous connaissons bien entendu le menu déroulant, célèbre dans les interfaces à deux dimensions.

Nous allons dans cette partie détailler les différentes métaphores de menu trouvée dans la littérature de réalité virtuelle.

Le Menu TULIP proposé par Bowman et Wingrave[BW01], offre un menu déroulant articulé autour des doigts de l'utilisateur. Chaque élément du menu est affiché en regard d'un doigt de la main de l'utilisateur, équipé d'un gant de données sensible au contact. La sélection se fait alors en pinçant le doigt correspondant à l'élément voulu avec le pouce. Si le menu comporte trop d'éléments, il est possible soit d'utiliser deux mains, soit de cascader les menus en menus et sous-menus. La figure 1.24 montre une utilisation de cette technique. L'intérêt de cette technique est de permettre une sélection très précise et rapide, tirant profit de l'agilité des doigts. Par contre, la technique n'est pas adaptée à des menus de taille importante. De plus, l'affichage se faisant à proximité de la main, celle-ci risque d'occulter le menu dans un environnement basé sur de la projection sur grands (figure 1.25) écrans comme les CAVE et Workbench, ce qui limite son utilisation aux environnements totalement immersifs dans lesquels la main est représentée par un avatar.

[23] La proprioception est la faculté de connaître la position de ses membres sans les regarder grâce aux tensions musculaires.

On pourrait envisager de décaler légèrement la représentation, mais ceci annulerait la corrélation visuelle immédiate entre un élément du menu et le doigt associé, rendant la métaphore trop fastidieuse à utiliser.

Cette métaphore est intéressante car elle exploite pleinement le côté proprioceptif de la main : nul besoin de regarder ses doigts pour savoir lequel va se plier : une fois que les menus sont connus, la sélection peut se faire en aveugle, comme une suite de plis de doigts, du moins tant que le nombre de combinaisons reste suffisamment faible.

Le Command and Control Cube proposé par Grosjean et al. [GC01], appartient également aux métaphores de contrôle d'application. En partant du concept des *Marking menus*[KB94], qui propose de disposer les menus en camemberts plutôt qu'en liste pour en accélérer l'accès , les auteurs proposent un menu composé de 27 boites réparties en 3x3x3 boites, qu'ils nomment le *Command and Control Cube*, ou C^3. Ce menu est illustré par la figure 1.26. L'idée sous-jacente est, comme pour les menus camemberts, de permettre à l'utilisateur de sélectionner une commande en faisant simplement un geste dans la bonne direction, peu importe son amplitude. Pour les utilisateurs entraînés, la manipulation peut devenir très efficace, puisque l'amplitude du geste n'est pas prise en compte comme elle l'est lorsqu'il faut atteindre une position donnée dans l'espace. Le C^3 se présente à l'utilisateur comme un sorte de rubix cube. 26 des 27 boites disponibles peuvent contenir une commande, la dernière –celle du centre– étant réservée à l'action spéciale d'annuler le menu sans effectuer la moindre action. A l'intérieur du menu, un petit curseur symbolise la position de la main de l'utilisateur, toujours initialisée au centre du cube. Ainsi, l'utilisateur n'a qu'à effectuer un geste dans la direction de l'élément souhaité, puis relâcher le bouton pour effectuer l'action correspondante. Les auteurs proposent de plus de tenir compte de la direction du regard par rapport à la position du menu afin de décider s'il est nécessaire ou non de l'afficher. D'après les tests menés par les auteurs, la métaphore donne d'excellents résultats en terme de rapidité de sélection, et ne génère que peu de sélections erronées.

Là encore, la métaphore sait tirer profit du sens proprioceptif. Nul besoin de voir sa main pour effectuer un déplacement vers la droite, surtout si son amplitude n'a aucune importance. Grâce à cet aspect, les utilisateurs entraînés peuvent manipuler sans retour visuel de manière très efficace, la métaphore jouant alors le rôle de «raccourci clavier».

Les menus circulaires ont été utilisés à plusieurs reprises. L'idée est ici de disposer les éléments sur un cercle, la sélection se faisant soit par mouvement dans la direction d'un élément à partir du centre, soit par rotation du disque de manière à ammener l'élément d'intérêt dans la zone de sélection. Le concept n'est pas propre à la réalité virtuelle, mais a été emprunté aux interfaces 2D. Les travaux de [KB94] mettent en évidence le gain de performance que peut procurer la disposition circulaire –en particulier leur possibilité de faire une sélection en ne faisant qu'un trait

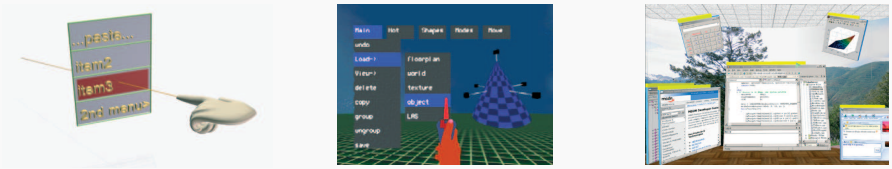


Illustration d'après [je92] — Image de [bow99] — Image du projet Infinite-3D (Rice University)

FIG. 1.22 – Menu 2D Converti

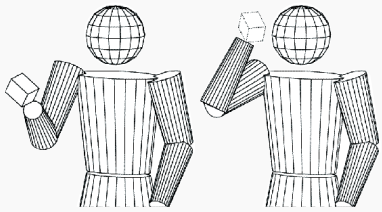


FIG. 1.23 – Suppression et restauration d'un objet tirant profit de la proprioception (image de [MJS97])

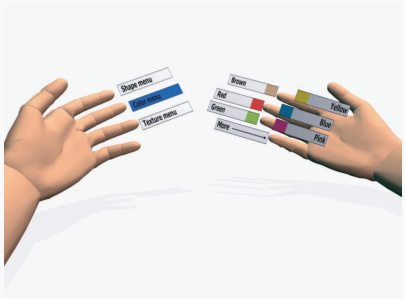


FIG. 1.24 – Le menu TULIP (image d'après [BW01])

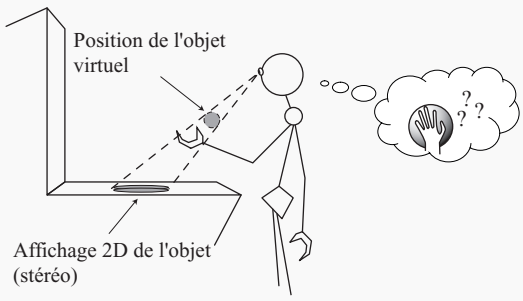


FIG. 1.25 – Le problème d'occlusion du à la main

dans la direction de l'élément considéré. On peut cependant noter une limitation sur la taille maximale de ces menus, limitation inhérante à la disposition géométrique des éléments : les éléments sont disposés en quadrants ou octants afin de ne pas nuire à la précision de la sélection.

L'adaptation du concept aux environnements virtuels fut introduite par Callahan et al. par les *menu tarte*[24] (figure 1.27). Leurs éléments sont équitablement réparties autour d'un point centre. La manipulation est ainsi plus efficace par rapport à menu linéaire, car tous les éléments sont équidistants du centre sur lequel s'ouvre le menu. Une amélioration notable de ces menus a été apportée par Deering [Dee95], en adaptant les menus pour être hiérarchiques et permettre ainsi un nombre plus important de choix (figure 1.27). L'inconvénient de ce type de menu est la place qu'ils occupent à l'écran, à tel point que la version hiérarchique remplace intégralement la scène durant la manipulation du menu.

Liang et al. utilisent aussi des menus circulaires, sans hiérarchie [LG94]. Ici, les éléments du menu sont disposés en cercle autour d'un axe vertical. L'élément faisant face à l'utilisateur est l'élément actif. Il est visuellement distingué par un trou dans l'anneau qui contient les autres éléments. L'utilisateur, pour changer l'élément actif, doit faire tourner la roue (ou les éléments), en pivotant son poignet selon un axe prédéfini. Liang utilise un périphérique aujourd'hui disparu, *the Bat*, pour tourner le menu. On regrette cependant l'absence d'évaluation et d'approfondissement de leur technique, dont ils ne parlent que succintement.

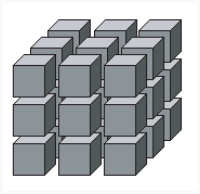
Droske et al. [WD00] proposent un menu à mi-chemin entre menu circulaire et sélection par pointage (figure 1.28). Dans leur implantation, chaque élément est symbolisé par une petite icône. Elles sont disposées devant l'utilisateur selon une portion d'arc de cercle. La sélection cependant se fait à l'aide d'un rayon laser, et s'apparente de ce point de vue plus à un paradigme 2D$\frac{1}{2}$ qu'à un paradigme 3D. Notons tout de même que le rayon est contraint à rester dans l'espace occupé par les éléments, évitant ainsi de sélectionner un élément ne faisant pas partie du menu. Malheureusement, les auteurs ne donnent que peu de détails sur les spécificités ou les performances de leur menu.

1.4 Conclusion

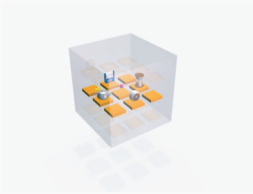
Les différents aspects matériels et logiciels de la réalité virtuelle que nous venons de voir sont loin d'être figées. La discipline est jeune, et connait des avancées régulières, autant sur le plan matériel que logiciel. Si la réalité virtuelle se bornait principalement il y a quelques années à de la visualisation d'objets que l'utilisateur pouvait tourner pour en observer l'intégralité, l'utilisateur peut aujourd'hui faire des tâches complexes comme planifier la réalisation d'un puits de pétrole [Dor04] ou recombiner des molécules [KvHPB02].

Tous ces progrès sont certes très stimulants, mais malgré tout, bon nombre de problèmes restent au gout du jour. On peut penser aux limitations du matériel, qui

[24] Pie menu

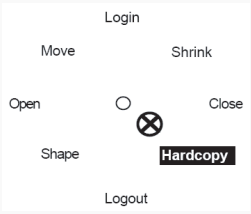


Subdivision en 3x3x3 cubes

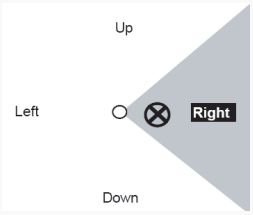


Affichage d'icônes dans le cube

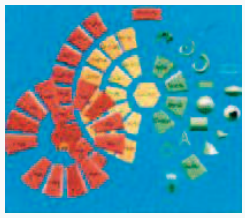
FIG. 1.26 – Le Command and Control Cube, ou C^3 (Figures d'après [GC01])



Menu circulaire 2D.



Menu circulaire 2D (zones visibles)



Menu tarte hiérarchique 3D.

FIG. 1.27 – Le menu tarte et sa version hiérarchique (images de [Dee95])

FIG. 1.28 – Le menu arrondi de Droske (image de [WD00])

en rendant la réalité virtuelle possible, en fixe également les limites. Mais le matériel évolue, et les limites sont repoussées. Les limitations proviennent également de l'usage que nous pensons à faire des environnements virtuels. L'élaboration d'un paradigme d'interaction évolué est une tâche longue et complexe. Mais malgré tout, nous sommes passés de méthodes relativement simples comme la désignation par rayon ou les menus 2D convertis à des paradigmes très nettement mieux adaptés comme HOMER ou le Command & Control Cube. Le but recherché est toujours de procurer à l'utilisateur les paradigmes les plus intuitifs, les plus efficaces et les plus agréables. Il est heureusement plus que probable que les progrès ne s'arrêtent pas en si bon chemin. La réalité virtuelle de demain devrait prendre de plus en plus d'importance et interesser un public de plus en plus large. Une avancée dans cette direction est faite en psychiatrie, où la réalité virtuelle peut par exemple aider à soigner une agoraphobie[25] en plaçant le patient dans un environnement urbain parfaitement contrôlé.

La réalité virtuelle intéresse, séduit, fascine, interroge... et ce n'est que le début.

[25] Peur de la foule.

Chapitre 2

Déformations de forme libre

2.1 Introduction

Les déformations de forme libre sont, depuis leur formalisation par Barr [Bar84], un outil classique de la modélisation géométrique, présents dans la majorité des logiciels du commerce. Il propose une formalisation des déformations telles que le cisaillement, l'écrasement, etc. indépendante de la représentation ou de la topologie des objets considérés. Sederberg et al. [SP86] reprit l'idée générale des travaux de Barr mais en la présentant sous un angle totalement différent. Pour lui la déformation consistait à déformer une portion de l'espce par le déplacement de quelques points clés, en appliquant une déformation similaire à tout ce qui se trouvait être dans la portion d'espace affectée. Ces déformations ont reçu le nom générique de FFD, pour *Free Form Deformation*. On ne se soucie guère de la forme initiale de l'objet, ce qui importe est de pouvoir associer un déplacement à chaque point de l'espace, que l'utilisateur peut contrôler à l'aide de différents moyens. Ces moyens divisent les FFDs en deux grandes familles : avec ou sans maillage de contrôle.

Nous verrons dans un premier temps les principales déformations à maillage de contrôle. Nous verrons ensuite les déformations contraintes, et le modèle de déformation DOGME en particulier.

2.2 Déformation avec maillage de contrôle

Les FFD proposées par Sederberg [SP86] reposent sur l'utilisation d'un maillage de contrôle (*lattice* en anglais), donnant la forme de la déformation. Son idée est d'englober l'objet (ou la partie de l'objet) à déformer dans un maillage de contrôle parallélépipédique, comprenant un nombre réduit de points. Cette étape est appelée *phase d'association*. Durant cette phase, chaque point $p =< x,y,z >$ de l'objet est associé à un point $p_a < u,v,w >$ de l'espace des paramètres du maillage de contrôle. Lorsque un objet et un maillage de contrôle sont associés, tout déplacement d'un point du maillage de contrôle donne lieu à un déplacement similaire sur un ou plusieurs points de l'objet associé. En effet, l'ensemble des points as-

sociés conservent leurs coordonnées p_a, et leurs coordonnées p sont recalculées à partir de p_a.

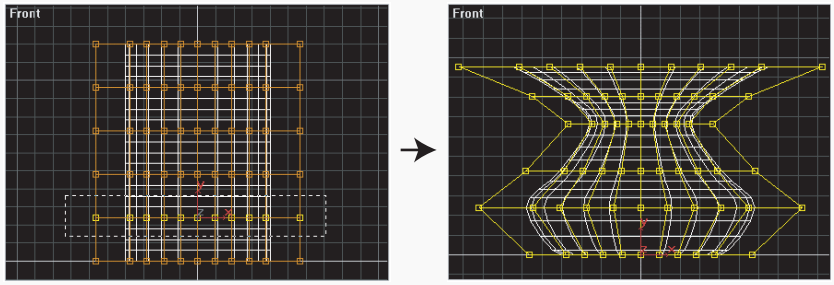
Initialement, le maillage de contrôle était un maillage de Bézier, mais on trouve également des versions utilisant des volumes Spline ou NURBS[LW94]. Bien entendu, l'utilité des FFDs repose sur le fait que le maillage de contrôle comporte un nombre très réduit de points par rapport à l'objet associé, et se manipule donc plus aisément que l'objet. Lorsque la forme souhaitée est atteinte, le maillage de contrôle n'a plus d'utilité et est désolidarisé de l'objet, qui bien sur conserve sa forme, et termine la déformation. La figure 2.1 montre quelques déformations classiques obtenues par cette technique.

La phase d'association requiert, quel que soit le type de maillage de contrôle, la capacité d'associer chacun des points à un point dans l'espace des paramètres du maillage de contrôle. Dans le cas d'un volume à trois dimensions (Bézier, Spline ou autres), cette phase est non ambiguë –si le maillage de contrôle ne comporte pas de singularités. Dans le cas d'une association avec une courbe, il peut arriver qu'un point puisse être associé à plusieurs paramètres (par exemple, le centre d'un arc de cercle).

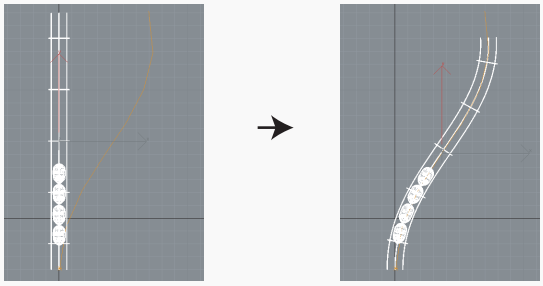
De nombreuses améliorations ont été proposées à ce modèle, principalement concernant l'utilisation de maillages de contrôle de forme non parallélépipédiques. En effet, plus la forme du maillage de contrôle s'éloigne de celle de l'objet associé, plus le parallèle entre les deux devient difficile à faire, et la déformation recherchée devient de moins en moins intuitive à obtenir. Coquillart propose des maillages de forme variées, allant du cylindre à l'étoile [Coq90]. D'autres formes de maillages ont été proposés, de dimension topologique variables, permettant des déformations différentes et originales. Des maillages de forme quelconque peuvent être utilisés grâce aux travaux de MacCracken [Mac96]. Bechmann et al. [BBT96] proposent également des maillages de contrôle de forme quelconque, en les représentant comme des ensembles de tétraèdres, rendant ainsi plus aisée la phase d'association entre l'outil et l'objet.

On peut noter les déformations axiales, comme les AxDf [LCJ93] ou Wires [SF98], dont l'outil est un simple axe qui traverse l'objet à déformer. Des outils surfaciques ont également été imaginés, comme les outils de [JF96], permettant de déformer l'objet associé en influant sur son profil. Elkouhen propose une approche différente de la déformation [Elk02], en se passant de la phase d'association explicite. Il peut ainsi déformer en utilisant un outil ponctuel, linéaire, surfacique ou volumique, auxquels des poignées contrôlant taille et torsion sont ajoutées. Ces différents modèles peuvent être facilement mélangés entre eux, comme le montre Peyré [PB02]. Enfin, notons les travaux de Milliron et al. [MJBF02], proposant une formulation mathématique englobant la quasi-totalité des déformations de forme libre sous un formalisme unique.

D'autres apports ont été proposés, notamment en ce qui concerne la manipulation du maillage de contrôle. Traditionnellement, le maillage (ou une partie) est modifié (par translation, rotation ou tout autre moyen), et l'utilisateur constate la déformation occasionnée. Il peut ainsi, par un processus itératif d'essais/erreurs,



Déformation d'un cylindre en vase par EFFD cylindrique.

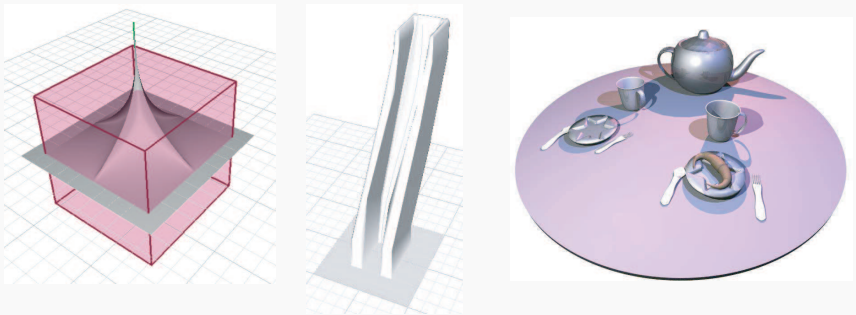


Déformations de rails par déformation axiale

En haut Transformation d'un cylindre en vase par application d'une EFFD de forme cylindrique (3D Studio MAX).

En bas Déformation de rails le long d'une courbe par déformation axiale (3D Studio MAX).

FIG. 2.1 – Applications classiques des déformations de forme libre



Volume d'influence cubique.

Volume d'influence voxel, contrainte curviligne.

Multiples objets simples déformés par des contraintes aux volumes d'influence variés.

FIG. 2.2 – Quelques déformations classiques obtenues avec DOGME (images de [Ger01]).

converger jusqu'à la déformation qu'il souhaitait au départ. Ce processus peut néanmoins être long et fastidieux. C'est pourquoi Hsu et al. proposé les DFFD [HHK92], pour *Direct-FFD*, ou FFD à manipulation directe. Dans ce modèle, l'utilisateur peut manipuler directement un point de l'objet associé ; les déplacements des points du maillage de contrôle sont automatiquement calculés pour satisfaire le déplacement. L'utilisateur bénéficie alors d'un contrôle direct sur la déformation, tout en n'ayant pas besoin de déplacer chacun des points de l'objet. Les DFFDs sont d'ailleurs, en ce sens, relativement proche de la seconde famille de déformations de forme libre faisant l'objet du paragraphe suivant.

2.3 Déformations sous contrainte

Contrairement aux FFDs que nous venons de décrire, les déformations sous contraintes ne reposent pas sur la manipulation d'un maillage englobant. Elles proposent plutôt de laisser à l'utilisateur le soin de déterminer exactement la déformation des points qu'il juge important de l'objet, et de laisser le modèle satisfaire les contraintes imposées. L'utilisateur peut contrôler la zone de l'espace affectée par chaque contrainte qu'il pose ainsi que d'autres paramètres dépendants du modèle. Cette famille de modèles met en avant la précision de la déformation obtenue, qui respecte exactement les contraintes spécifiées par l'utilisateur, contrairement aux FFD à maillage qui favorisent d'avantage l'aspect interactif de la manipulation.

2.3.1 Dogme

Le premier modèle de déformation géométrique sous contrainte a été élaboré en 1991 par Borrel et Bechmann[BB91]. Ce modèle a été baptisé DOGME, pour *Deformation Of Geometric Model Editor*. Il permet de spécifier un nombre arbitraire de contraintes à appliquer à l'espace dans lequel sont plongés les objets devant être déformés. Ces contraintes sont originalement définies comme un couple de point : un point de départ et un point d'arrivée, définissant un segment de droite. La déformation consiste à faire en sorte qu'après application, les points de départ de toutes les contraintes soient placés exactement à leur point d'arrivée respectifs. Une certaine portion de l'espace est affectée par chaque contrainte, selon la formulation d'une fonction mathématique appelée *fonction d'extrusion*, associée à chaque contrainte. On peut ainsi agir soit sur tout l'espace, soit sur une portion réduite uniquement, nommée *volume d'influence*. Le modèle permet ainsi de représenter aussi bien les déformations à support global (affectant tout l'espace, comme les translations) et à support local.

Plusieurs travaux ont fait suite à ce modèle. On peut noter les travaux de Bechmann et Dubreuil [BD93], qui introduit des volumes d'influence parallélépipédiques convexes grâce à un repère local associé au point de contrainte. Aubert propose quant à lui d'exprimer la déformation en coordonnées polaires [AB97], ce qui change fondamentalement les effets obtenus. Enfin, notons que Brandel a spécifié

algébriquement l'ensemble du modèle [BBB98], garantissant ainsi sa l'exactitude du modèle par rapport à ses hypothèses.

Bechmann et al. ont étendu le concept à des contraintes courbes [BG03] et aux volumes d'influences de forme quelconque par l'utilisation de voxels. La figure 2.2 montre quelques déformations typiques obtenues à partir de ce modèle. Ce modèle de déformation présente un certain nombre de caractéristiques intéressantes :

Précision de la déformation La déformation est calculée de manière à respecter exactement les contraintes spécifiées par l'utilisateur, d'un point de vue numérique. En cas de conflit entre plusieurs contraintes, la déformation peut être choisie pour minimiser un certain critère.

Généricité Le modèle permet nativement de représenter toutes les transformations géométriques habituelles (rotation, translation, homothétie, symétrie, etc.), tout comme des déformations plus caractéristiques que l'on contrôle grâce à une *fonction d'extrusion*, qui donne la forme de la déformation dans l'espace environnant la contrainte. De plus, [BG03] ont montré que le modèle pouvait être étendu pour prendre en charge tout type de déformation pouvant s'écrire comme une fonction $f : R^n => R^n$ (en particulier les déformations à maillage de contrôle), ce qui offre un cadre mathématique unique pour l'ensemble des déformations de forme libre. Les auteurs illustrent cette propriété en formulant le modèle Wires [SF98] dans leur formalisme.

Déformations sous-contraintes Il est possible de sous-contraindre la déformation, de manière à pouvoir optimiser d'autres critères que la précision géométrique de la déformation. Par exemple, [FA97] a proposé une technique permettant, outre la satisfaction géométrique des contraintes, de minimiser le changement de volume que la déformation occasionne sur un objet.

2.3.2 Formulation

Le modèle repose sur l'inversion d'une matrice, représentant les contraintes à satisfaire. On va pour cela projeter l'espace source dans un espace temporaire de dimension supérieure, puis le re-projeter à nouveau dans l'espace de destination, selon une perspective qui satisfasse au mieux les contraintes. Cette opération de projection correspond à l'inversion de la matrice. Celle-ci n'est cependant pas toujours inversible, notamment dans le cas où deux contraintes agissent de manière différente sur une même zone de l'espace, ou lorsque la déformation est sous-contrainte. On a dans ce cas recours à la pseudo-inverse [BO71], qui permet d'obtenir une déformation visuellement satisfaisante (quoique ne respectant pas nécessairement toutes les contraintes). La pseudo-inverse M^+ est une valeur approchée de l'inverse d'une matrice $Mm \times n$ lorsque celle-ci n'est pas inversible. La pseudo-inverse vérifie les propriétés 2.1 à 2.4.

$$MM^+ = M \tag{2.1}$$

$$M^+MM^+ = M^+ \tag{2.2}$$

$$(MM^+)^T = MM^+ \tag{2.3}$$
$$(M^+M)^T = M^+M \tag{2.4}$$

Il est également intéressant de noter qu'étant donné le problème $Mz = c$, alors $z = M^+c$ est la meilleure solution au problème au sens des moindres carrés.

Les étapes de l'application de la déformation sont décrites dans les équations 2.6 à 2.10, issues de [BG03], à partir de l'équation générale de la déformation en 2.5.

$$\underbrace{d(U)}_{n*1} = \underbrace{M}_{n*m}\underbrace{f(U)}_{m*1} \tag{2.5}$$

Où U représente le point auquel la déformation est appliquée, $d(U)$ le déplacement appliqué à U, M la matrice de projection calculée pour satisfaore les contraintes et $f(U)$ la fonction d'extrusion.

Chaque contrainte est représentée par un point de départ, noté V_i, ainsi qu'un vecteur de déplacement, noté dV_i. Dans l'équation 2.5, la seule inconnue est la matrice M. C'est donc elle que nous cherchons à calculer, de sorte que chaque contrainte soit respectée.

Afin d'autoriser des contraintes dont le chemin est courbe, nous allons commencer par éclater l'équation 2.5 selon chaque dimension de l'espace considéré. Ainsi, pour chaque points affecté par la déformation, son vecteur de déformation $d(U)$ pourra être non colinéaire au vecteur $d(U')$ d'un point U' voisin, affecté par la même contrainte. On peut ainsi à partir de vecteur de déplacement dV_i d'une contrainte, exprimer le déplacement des points avoisinants comme des déplacements dépendants mais non colinéaires à dV_i, pour former des chemins courbes[1]. La déformation d'une contrainte particulière sur un point U donné s'écrit alors (en 3 dimensions, mais valable en nD également) :

$$\begin{cases} d_x(U) = \alpha(U) \cdot dV_x \\ d_y(U) = \beta(U) \cdot dV_y \\ d_z(U) = \gamma(U) \cdot dV_z \end{cases} \tag{2.6}$$

avec $\alpha(U), \beta(U)$ et $\gamma(U)$ sont des fonctions de paramétrisation qui définissent le chemin de la déformation. En regroupant l'ensemble des contraintes au sein d'un

[1] Cette formulation peut poser problème lorsqu'une ou plusieurs composantes du vecteur dV sont nulles. Tous les vecteurs obtenus par multiplication de dV conserveront la nullité de ces composantes, et empêchent la déformation de suivre exactement le chemin spécifié. Une solution facile à mettre en oeuvre –quoique imparfaite– est de détecter ce type de cas de figure et de «forcer» dV_x à une valeur infinitésimalement petite. Une meilleure solution consiste à exprimer le vecteur dV dans un repère dans lequel aucune de ses coordonnées n'est nulle, par une matrice K : $d_x(U) = \alpha \cdot K \cdot dV_x \cdot K^{-1}$.

système, on obtient la formulation 2.7 :

$$\begin{cases} d_x(U) = M f_x(U) \\ d_y(U) = M f_y(U) \\ d_z(U) = M f_z(U) \end{cases} \tag{2.7}$$

Le calcul des matrices M se fait de la manière suivante : Les n_c contraintes, remplacées dans l'équation 2.7 définissent un système de $n * n_c$ équations :

$$\begin{cases} d_x(V^i) = M f_x^i(V^i) \\ d_y(V^i) = M f_y^i(V^i) \\ d_z(V^i) = M f_z^i(V^i) \end{cases} \quad \forall i \in [1, n_c] \tag{2.8}$$

En regroupant ces équations en 3 vecteurs, D_x, D_y et D_z on peut réécrire ce système de la manière suivante grâce à la transposition :

$$\begin{cases} D_x = X_x M_x^t \\ D_y = X_y M_y^t \\ D_z = X_z M_z^t \end{cases} \tag{2.9}$$

avec $D_x = \begin{pmatrix} d_x^t(V_1) \\ \cdots \\ d_x^t(V_{n_c}) \end{pmatrix}$, $X_x = \begin{pmatrix} f_x^t(V_1) \\ \cdots \\ f_x^t(V_{n_c}) \end{pmatrix}$ et similairement pour D_y, D_z, X_y, X_z.

Il s'agit alors de résoudre le système 2.9, ce qui est fait de la manière suivante :

$$M_x^t = X_x^+ D_x + (I - X_x^+ X_x)\xi_x \tag{2.10}$$

où X_x^+ représente la pseudo-inverse de X_x et ξ_x un vecteur choisi arbitrairement. La résolution pour M_y et M_z se fait similairement. On déduit alors aisément les matrices M à partir de leur transposée. La résolution du système ainsi que les propriétés des solutions sont larement discutées aux pages 8 à 10 de [Bec95]. La seconde partie de l'équation 2.10 permet, dans le cas de systèmes sous-contraints, d'optimiser d'autres critères en plus de la précision géométrique de la déformation, comme par la minimisation du changement de volume occasionné par ladite déformation [FA97], en navigant dans l'ensemble des déformations satisfaisant les contraintes.

2.4 Paramétrisation

Notons qu'il est toujours possible de composer les fonctions d'extrusion avec des fonctions de paramétrisation, placées avant ou après f, comme dans l'équation 2.11.

$$d(U) = M(h(f(g(U))) \tag{2.11}$$

où f est la fonction d'extrusion et g et h des fonctions de paramétrisation. De telles fonctions permettent d'apporter d'avantage de généricité au modèle, en permettant par exemple la composition de volumes d'influence.

2.5 Scodefs

Le modèle de déformations DOGME a été repris peu après sa création pour aboutir à un modèle similaire et légèrement simplifié, ne prenant en compte que le cas particulier des systèmes *bien contraints*, c'est à dire ni sous ni sur-contraints, et en 3D : le modèle *Scodef* [BR94]. La formulation de ce modèle est de ce fait relativement proche de celui de[BB91]. Ce modèle a été étendu par Raffin [RN99, Rom00]. Il étend d'une part la forme des contraintes à des courbes, ainsi que l'utilisation de volumes d'influences super-quadriques.

2.6 Maillages adaptatifs

L'un des atouts, mais également l'un des défauts, des déformations de forme libre -toutes catégories confondues- est leur absence d'hypothèse quant à la représentation des objets déformés. Cela permet bien sur de traiter tout types d'objets, mais impose également que la représentation topologique de l'objet ne change pas au cours de la déformation. Ceci peut poser de nombreux problèmes, notamment dans les zones de forte déformation : l'objet de départ peut s'avérer mal échantillonné (en nombre de triangles ou toute autre primitive) pour représenter fidèlement la déformation souhaitée. Nombre de propositions ont été faites pour permettre d'adapter un maillage selon les besoins sous l'effet d'une FFD, comme [GD99] qui se servent directement de la déformation pour savoir ou ajouter ou supprimer des triangles au besoin, adaptant ainsi le maillage à la situation. La figure 2.3 illustre le processus de cet algorithme. En optant pour d'autres représentations pour les objets, comme des surfaces de subdivision ou des surfaces paramétriques, la phase d'adaptation du maillage peut être plus automatique puisqu'elle fait partie intégrante de l'affichage même de ces modèles.

2.7 Comparatif et conclusion

Les modèles de déformations de forme libre ont un certain nombre de points communs. Ils adressent souvent le même problème sous une optique différente, ce qui leur donne à chacun certaines spécificités que les autres modèles n'auront pas. Nous allons dans ce paragraphe donner une classification de principaux modèles de déformation selon quelques critères simples :

Support définit la zone de l'espace à laquelle la déformation peut s'appliquer. Un support *global* fait intervenir tout l'espace (la rotation ou l'homothétie

sont considérées comme globales, malgré leur centre restant immobile), un support *local* laisse certains points inchangés.

Précision quantifie la précision avec laquelle une certaine déformation est obtenue. *Approx.* définit une précision approximative, c'est à dire que l'utilisateur doit procéder par essais/erreurs pour aboutir à la forme de déformation qu'il souhaite. *Précis* indique que le modèle est intrinsèquement écrit pour permettre une expression explicite de la déformation souhaitée, et l'atteint en une seule étape.

Contrôle de la forme permet de décider si l'utilisateur peut spécifier comment la déformation est transmise à l'objet.

Coût par point donne une indication de la complexité de la méthode de déformation, pour chaque point déformé. La colonne *Association* donne un ordre de grandeur de la phase d'association de l'outil de déformation avec l'objet, la colonne *Déformation* donne l'ordre de grandeur pour l'application de la déformation à l'objet.

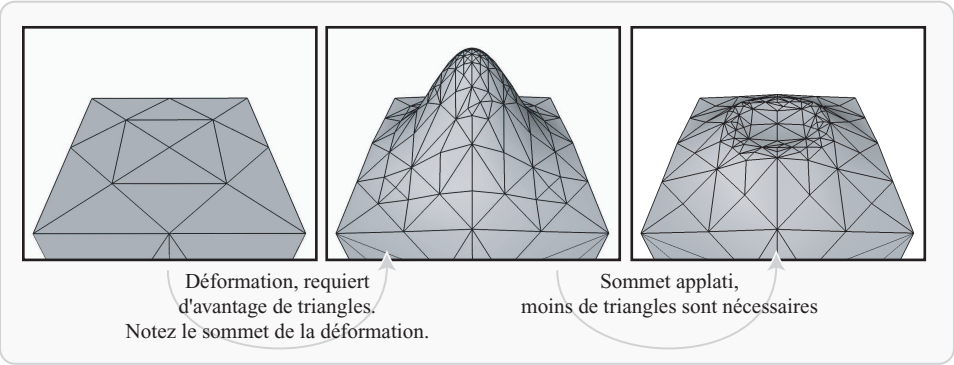


FIG. 2.3 – Adaptation d'un maillage à une déformation (Figure d'après [GD99])

Modèle	Support		Précision		Contrôle de la forme	Association	Déformation
	Global	Local	Approx.	Précis			
FFD		✓	✓	⊕	⊕	constant	O (type de volume)
AxDf		✓		✓	⊕	constant	constant
Wires		✓		✓	×	constant	constant
Dogme	✓	✓		✓	✓	O (nombre de contraintes)	O (nombre de contraintes)
Scodef	✓	✓		✓	✓	O (nombre de contraintes)	O (nombre de contraintes)

Légende

✓	Pris en charge automatiquement
⊕	Non pris en charge automatiquement, mais des extensions existent.
×	Non supporté

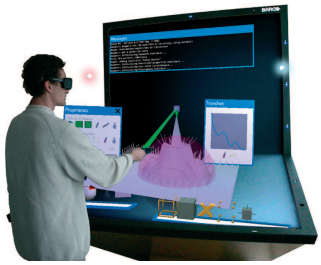
TAB. 2.1 – Comparatif des différents types de déformations de forme libre

Deuxième partie

Travail personnel

Chapitre 3

Déformation d'objets et Réalité Virtuelle



3.1 Introduction

Dogme^{RV} est le prototype qui a servi de centre d'essais aux diverses métaphores et techniques d'interactions testées et implantées. Il s'agit d'un micro-modeleur géométrique basé sur les déformations de forme libre sous contrainte, et plus particulièrement du modèle de déformations DOGME (d'où son nom). Il permet à un utilisateur d'appliquer des contraintes de déplacement dans un espace où sont plongés des objets géométriques. L'utilisateur doit avoir toute liberté quant à la manipulation des objets : l'ensemble de l'environnement dans lequel il se trouve doit être réactif et pouvoir être manipulé, que ce soient les contraintes elles-mêmes, leurs nombreuses propriétés ou les objets.

Nous allons détailler dans ce chapitre les différentes données manipulées et les techniques d'interaction permettant de les modifier. Les interactions possibles dans le prototype se situent à deux niveaux logiques : l'utilisateur peut manipuler les données qui ont directement trait au problème traité –contrainte, objets, etc., et les données qui ont trait à l'application elle-même –palettes d'outils, boutons de

commande, etc. Dans un premier temps, nous détaillerons la manipulation des données (objets, contraintes et leurs attributs tels que fonction d'extrusion et volume d'influence). Nous détaillerons ensuite la partie contrôle, sous trois métaphores différentes. Nous terminerons le chapitre par un test utilisateur et son analyse.

3.2 Dogme^{RV}

Le logiciel Dogme^{RV}, faisant suite à un travail de DEA sur l'extension du modèle de déformations sous-jacent, est avant tout un centre d'essais de manipulation et d'interaction 3D sur le plan de travail virtuel. Les déformations de forme libre nous ont semblé être une application pouvant tirer grand avantage de la réalité virtuelle pour plusieurs raisons, dont on peut en retenir deux principales.

Tout d'abord, les déformations de forme libre sont définies dans un contexte de modélisation géométrique 3D, et profitent donc pleinement de l'interaction elle-même tridimensionnelle. Contrairement à une manipulation classique, où l'on est contraint à passer par une phase de projection en 2D pour l'affichage (et implicitement la manipulation, ce qui est une limite contraignante), la réalité virtuelle et ses périphériques spécifiques nous permettent une manipulation 3*D* de données 3*D*, en superposant l'espace des données et l'espace de manipulation.

Par ailleurs, les données manipulées dans le cadre de la modélisation géométrique sont abstraites, et ouvrent donc toute latitude pour des techniques d'interaction non bornées à l'imitation de la réalité.

Le prototype a été élaboré sur un plan de travail virtuel à deux écrans. Il peut être utilisé sur une configuration mono-écran, mais ne se prêterait pas bien à des environnements de type CAVE, à cause de la disposition des objets dans la scène.

Dans ce prototype, l'utilisateur est amené à manipuler principalement des données abstraites, comme des points de contrainte, des volumes d'influence ou des fonctions d'extrusion. Chacun de ces objets dispose d'un certain nombre d'éléments réactifs à la manipulation, et chacun permet un jeu précis d'interactions. Il y a principalement trois familles d'objets que l'utilisateur sera amené à manipuler :

Les contraintes forment le coeur du logiciel, et sera la plus fréquemment manipulée. Les contraintes sont des objets composites, c'est à dire qu'elles sont formées elles-mêmes de sous-objets réactifs à la manipulation. Nous reviendrons en détail sur ceci.

Les objets permettent de visualiser le résultat de la déformation. Ils sont donc déplaçables afin de pouvoir les placer où cela est nécessaire pour visualiser la déformation ou créer une scène à partir d'objets et de déformations.

Les contrôles permettent de manipuler l'application elle-même. L'utilisateur les utilise fréquemment, pour changer de mode d'interaction ou créer une nouvelle contrainte par exemple.

Ces trois familles d'objets réactifs peuvent être groupés en deux ensembles logiquement liés : les contraintes et les objets forment les données d'application,

alors que le contrôle forme un groupe à part. La différence entre les deux groupes provient du fait que quelle que soit l'application considérée, son contrôle peut rester similaire, alors que les données d'application vont changer du tout au tout d'une application à l'autre. Les prochaines parties décrivent les interactions possibles sur les différents objets du prototype.

3.3 Sélection

Nous avons déja défini la notion de paradigme d'interaction (section 1.1.2, page 9) comme étant le moyen donné à l'utilisateur de communiquer avec son environnement virtuel. Dans Dogme^{RV}, nous avons essayé de centrer au maximum l'application autour du paradigme de *rayon laser virtuel*, qui nous est apparue comme la plus adaptée à la fois au matériel et à l'application considérée. Ne disposant pas de gants de données, les métaphores basées sur la préhension nous ont semblées moins adaptées, puisque l'outil matériel d'intéraction ne permettait pas de manipuler «à main nue», ni de générer un avatar convainquant de la main de l'utilisateur.

Les inconvénients connus de la métaphore du laser, notamment les problèmes de sélection d'objets lointains, n'entrent que peu en compte dans notre application, étant donné qu'elle est destinée à manipuler des objets virtuellement posés sur le plan de travail virtuel, toujours à une distance raisonnable et de taille réduite.

La métaphore du rayon laser, telle qu'elle est implantée dans le prototype, sera décrite plus amplement dans le Chapitre 5. D'un point de vue utilisateur, le rayon se présente comme un faisceau de lumière rouge partant de sa main (ou plus précisément de son périphérique d'interaction), similairement à une lampe torche au faisceau très resseré qu'il porterait. En réalité, l'appellation *rayon* est un peu abusive, car le rayon est doté d'une épaisseur paramétrable, et peut même devenir conique. Lorsque ce rayon intersecte un objet, cet objet passe dans un état *actif*, qui indique qu'il est candidat à la manipulation. Cet état est indiqué visuellement par un changement de teinte de l'objet. Si, alors qu'un objet est actif, l'utilisateur appuie sur un bouton du périphérique d'entrée, l'objet actif passe alors à l'état *manipulé*. Le type de manipulation appliquée à l'objet dépend de l'objet lui-même –un objet géométrique sera par exemple déplacé ou tourné, alors qu'un bouton d'action effectuera l'action qui lui correspond. L'objet reste dans cet état *manipulé* tant que le même bouton n'est pas relâché, peu importe les autres boutons pressés et relâchés entre temps.

A l'utilisation, cette technique d'interaction apparaît comme très facile à comprendre et utiliser, même pour des utilisateurs sans expérience préalable de la réalité virtuelle, ce qui nous semble être une qualité appréciable de la métaphore. On peut cependant noter que les objets doivent avoir une taille suffisante (de l'ordre du dm^3 dans notre configuration) pour être désignés efficacement.

3.4 Manipulation des objets et contraintes

3.4.1 Manipulation des objets

Par définition, un modèle tel que DOGME opère sur un espace entier peu importe ce qui s'y trouve. Afin de visualiser le résultat de la déformation, il est nécessaire de placer des marqueurs visibles dans cet espace, afin d'expliciter visuellement les modifications de l'espace de plongement. Toutes sortes de modes d'affichage sont envisageables, en allant d'un nuage de points à la visualisation du champ vectoriel de déformations. Nous avons retenu l'option de placer des objets géométriques passifs (jouant d'une certaine manière le rôle de marqueurs) dans cet espace déformé. Cette solution est la plus proche de l'utilisation commune des déformations de forme libre : appliquer une déformation à des objets géométriques. L'affichage du champ de déformation sous forme d'un nuage de droites reliant les points non déformés à leur image est également disponible à des fins d'expérimentation mais s'avère peu utile dans la pratique car difficile à interpréter.

Les objets marqueurs bénéficient d'une interaction très classique : l'utilisateur peut les déplacer (rotation et translation), et changer leur taille (homothétie). Il n'est pas possible d'éditer la géométrie interne de l'objet par ajout ou suppression de triangles ou générer un nouvel objet défini comme l'union de deux autres, etc. La seule modification qui puisse affecter un objet outre les transformations affines mentionnées précédemment est l'application définitive de la déformation à la position de ses points. Notons que cette opération n'affecte que le plongement de l'objet et non sa topologie.

Translation et rotation

Chaque objet dispose d'une poignée de translation/rotation, symbolisée par une petite sphère bleue. Seule cette sphère réagit à la métaphore du laser (et non l'objet entier), afin d'éviter de rendre insaisissable tout élément se trouvant à l'intérieur de l'objet –comme une contrainte, par exemple. Lorsque cette sphère est manipulée, les translations de la main de l'utilisateur sont transmises à l'objet, modulées par un facteur de distance augmentant la vitesse des objets lointains, et les rotations sont appliquées par rapport au centre géométrique de l'objet. Pour cette manipulation, la métaphore rejoint donc la méthode HOMER [BH97] (sélection par pointage et manipulation locale). Le facteur correctif que nous appliquons sur les translations est destiné à compenser la relative lenteur des translations (comparées à une manipulation centrée utilisateur). Ainsi, les objets lointains seront déplacés plus rapidement que les objets proches, donnant l'illusion que la métaphore «réagit mieux». La précision se retrouve bien entendu légèrement diminuée, sans être pénalisante dans notre cas particulier.

Homothétie

Chaque objet dispose de trois poignées supplémentaires permettant d'effectuer une homothétie selon l'un de ses axes locaux. Pour cela, l'utilisateur désigne à l'aide du laser quel axe il souhaite manipuler, et appuie sur le bouton d'action. A partir de ce moment, la distance d entre l'origine du repère et le point le plus proche du rayon le long de l'axe est mémorisée. Durant la manipulation, la distance d' entre l'origine de l'axe de manipulation et l'intersection du rayon avec l'axe est réévaluée à chaque mouvement. Une homothétie de facteur $r = \frac{d'}{d}$ est appliquée à l'objet (par rapport à sa taille d'origine). La manipulation prend fin lorsque le bouton est relâché. La Figure 3.2 page 62 illustre le fonctionnement de cette opération.

Là encore, la manipulation ressemble à celle proposée par de nombreux produits du commerce. Elle est facilement comprise par les utilisateurs, mais pose quelques problèmes de précision. En effet, plus d est faible –i.e. l'utilisateur a démarré la manipulation *près* de l'origine de l'axe– plus la manipulation sera sensible et imprécise. Pour compenser cet effet, nous pouvons jouer sur la portion d'axe qui peut initialiser la manipulation : si seule la partie centrale permet de démarrer l'homothétie, d sera automatiquement bornée de sorte que l'objet ne puisse pas avoir une taille inférieure à $1dm^3$, taille que nous avons jugée minimale pour rester facilement sélectionnée sur notre environnement.

3.4.2 Contraintes

Les contraintes, au sens défini dans le modèle de déformations DOGME, représentent une grande partie des éléments interactifs du prototype. Chaque contrainte est définie par plusieurs éléments, illustrés en Figure 3.3 page 62 :

- son point de départ (V) ;
- son point d'arrivée (Q), qui représente l'image du point V après application de la contrainte ;
- son chemin de déformation (K), qui donne le chemin que suit la déformation entre V et Q, contenant au minimum deux demi-tangentes (T) ;
- son volume d'influence (I), qui définit la portion d'espace affectée par la déformation
- sa fonction d'extrusion (F), qui définit la forme de la déformation à l'intérieur du volume d'influence I.

Chaque élément peut être manipulé par l'utilisateur, soit en influant directement sur l'élément lui-même, soit par l'intermédiaire du contrôle d'application dans le cas de notions abstraites comme la *fonction d'extrusion*. La figure 3.4 illustre la palette de propriétés d'une contrainte, telle qu'elle est affichée sur une fenêtre 3D. Sous d'autres modalités de contrôle d'application, cette palette est remplacée par un menu contextuel.

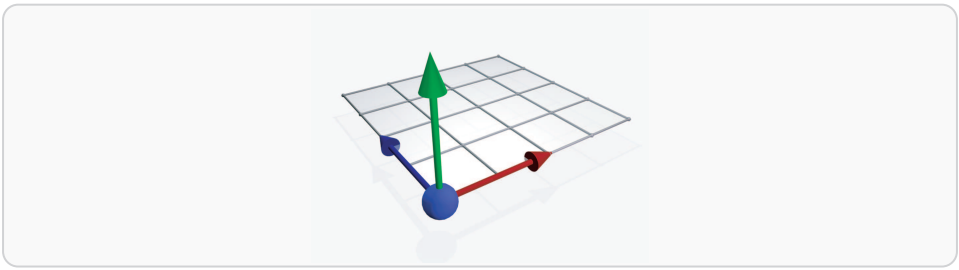


FIG. 3.1 – Poignées de manipulation d'un objet

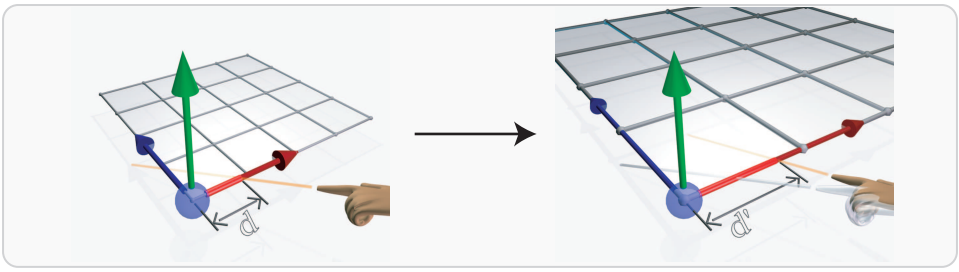
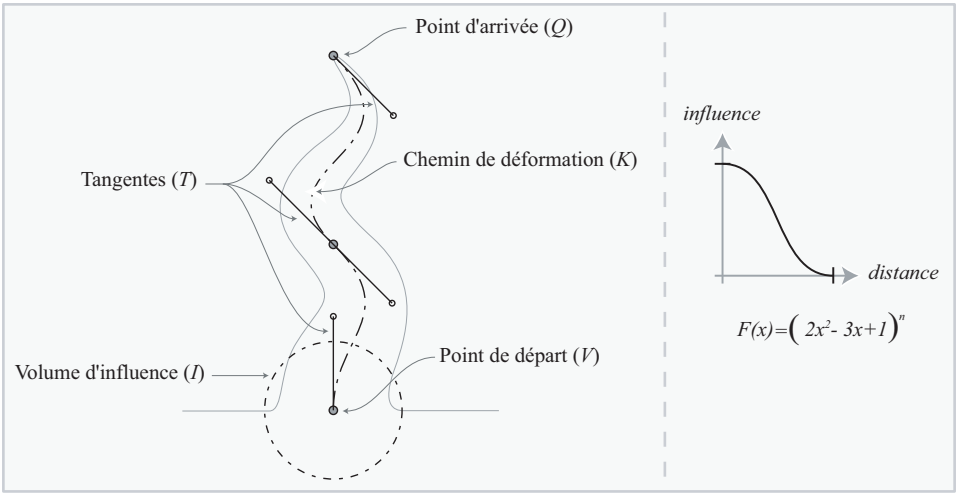


FIG. 3.2 – Fonctionnement de l'homothétie



A gauche : La contrainte. A droite : Une fonction d'extrusion à un paramètre (n).

V	Point de départ	Section 3.4.2
Q	Point d'arrivée	
K	Chemin de déformation	Section 3.4.2
T	Tangentes de manipulation	
I	Volume d'influence	Section 3.4.3
F	Fonction d'extrusion	

FIG. 3.3 – Éléments constituants d'une contrainte géométrique

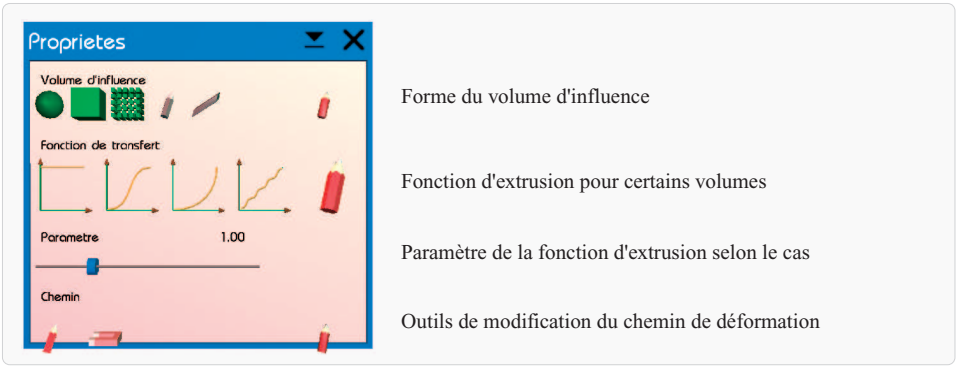


FIG. 3.4 – Palette de propriétés d'une contrainte

Points de contrainte

Une contrainte est par définition localisée en un certain point de l'espace. Celui-ci, ainsi que son point image, peuvent être déplacés librement. Les points V et Q sont symbolisés chacun par une petite sphère, que l'on peut déplacer à l'aide du rayon laser virtuel. La manipulation de ces éléments est *centrée sur l'utilisateur*, puisque les éléments sont des points infiniment petits qui n'ont pas d'orientation.

Chemin de déformation

Les deux points V et Q sont reliés par un chemin de déformation que l'utilisateur peut manipuler de plusieurs manières. Ce chemin est représenté comme une courbe de bézier cubique par morceaux, avec respect de la continuité géométrique (G_0).

Il est possible de créer rapidement une ébauche de courbe en laissant l'utilisateur la dessiner dans l'espace. Ce mode, activé par l'intermédiaire d'un menu, va enregistrer position et orientation de la main de l'utilisateur tant qu'un certain bouton est appuyé et qu'une distance minimale a été parcourue depuis le dernier échantillon enregistré. Ainsi, pour obtenir une forme particulière de courbe, il suffit de la dessiner en l'air à main levée. Les points échantillons ainsi obtenus sont ensuite convertis en courbe de Bézier cubique par morceaux en utilisant l'algorithme de Schneider[Gla90], afin de permettre d'autres types d'interactions, plus localisées. Cette technique permet d'obtenir rapidement une ébauche de courbe, que l'on peut ensuite affiner à l'aide des autres interactions disponibles. Elle n'est pas d'une précision extrême en soi (dessiner «en l'air» est un processus intrinsèquement approximatif), mais l'échantillonnage par distance plutôt que par pas de temps ainsi que la conversion en courbe de Bézier gomme les petits tremblements produits par l'utilisateur.

La représentation en Bézier cubique par morceaux permet une manipulation de type *points de passage et tangentes*, très répandue dans les logiciels de dessin

vectoriel. A chaque point de passage sont associées deux tangentes, qui définissent localement la courbure. Ces tangentes, ainsi que le point de passage, peuvent être déplacées ; la courbe est ainsi localement modifiée. La courbe étant G_1-continue, le déplacement d'une extrémité d'une tangente implique un déplacement de l'extrémité opposée –la courbe est G_0-continue si deux tangentes qui se suivent sont colinéaires. Ce type d'interaction permet d'ajuster de manière très précise et intuitive la forme de la courbe, avec un minimum d'entraînement. Il s'agit de plus d'une manipulation qui a un effet local sur la courbe, et touche au plus deux segments.

Une autre manipulation locale est proposée par l'intermédiaire des *aimants virtuels* : l'utilisateur tient en main un aimant répulsif ou attractif, qui respectivement repousse ou attire la courbe en fonction de sa distance à celle-ci. Le rayon d'influence de l'aimant est ajustable par un joystick présent sur le périphérique d'interaction : on peut ainsi effectuer aussi bien des modifications de grande envergure comme des modifications très localisées.

Pour les besoins de cette technique, la courbe de bézier est échantillonnée, l'opération magnétique s'appliquant sur les poins échantillons selon les équations illustrées en Figure 3.5 page 65. Notons que le traitement peut rapprocher ou éloigner les points échantillons les uns des autres. Afin de conserver une courbure et une apparence uniforme, des nouveaux points échantillons peuvent être introduits ou supprimés selon que leur densité augmente ou diminue. A la fin de la manipulation, la courbe est re-convertie en Bézier cubique par morceaux. Il est important de souligner que lors de la reconversion en Bézier cubiques, le nombre de segments créés peut être différent du nombre de segments que possédait la courbe avant manipulation afin de s'accommoder des points d'inflexion créés ou supprimés. L'algorithme général de ces outils est détaillé en Figure 3.6 page 65.

3.4.3 Volume d'influence

A chaque contrainte est associé un volume d'influence, pouvant être de différents types (paramétrable, explicite, discret), qui définit la portion d'espace affecté par la déformation ainsi que la «quantité» de déformation que reçoit chaque point contenu dans ce volume. Dogme^{RV} permet l'utilisation de volumes implicites et discrets, manipulés tous deux de la même manière à quelques exceptions près. Leur position est toujours attachée au point de départ V de la contrainte, il n'est donc pas possible de déplacer le volume d'influence directement. L'orientation et la taille peut quant à elle être modifiée à l'aide d'une poignée. Lorsqu'elle est manipulée, la poignée contrôle à la fois la rotation et la taille du volume d'influence. Cette manipulation est très simple, mais a le désavantage de contrôler deux grandeurs avec une seule poignée. Il est par conséquent difficile voire impossible d'obtenir certaines combinaisons *taille/position* particulières.

Nous avons donc également la possibilité de manipuler la taille et l'orientation indépendamment. L'utilisation de deux poignées séparées est nécessaire, l'une pour l'orientation autour du point de contrainte, l'autre (en réalité composée de trois poignées, une pour chaque dimension) pour la taille.

$$m(p,c,r) = \begin{cases} c & \text{si } ||x-c|| > r \\ c+(1+\alpha)(x-c) & \text{sinon} \end{cases} \quad (3.1)$$

avec :
- $p \in R^3$ le point à traiter
- $c \in R^3$ le centre de l'outil
- $r \in R$ le rayon de l'outil
- $\alpha \in [-1..1]$ intensité de l'attraction/répulsion (> 0 : répulsif, < 0 : attractif)

FIG. 3.5 – Equations de l'outil *aimants virtuels*.

```
// Algorithme de l'aimant virtuel sur une courbe échantillon

Fonction AimantVirtuel (
        P : Points échantillon ;
        x : Position de l'outil ;
        r : Rayon de l'outil
) : Points échantillon

Pour chaque point P(i) de la courbe Faire
    P'(i) ← m(P(i),x,r)
    d ← ||P'(i)-P'(i-1)||

    // Insersion de points si nécessaire
    TantQue d > d_max Faire
        Insérer un point m((P(i)+P(i-1))/2,x,r) entre P'(i) et P'(i-1)
        d ← ||P'(i)-P'(i-1)||
    Fin TantQue

    // Suppression de points si nécessaire
    TantQue d < d_min Faire
        Retirer le point P'(i)
        d ← ||P'(i)-P'(i-1)||
    Fin TantQue

Fin Pour

Fin Fonction
```

FIG. 3.6 – Algorithme des *aimants virtuels*.

3.4.4 Saisie de fonction d'extrusion

Certains types de volume d'influence nécessitent l'utilisation d'une *fonction d'extrusion* permettant d'associer une quantité de déformation à chaque point en fonction de leur distance avec le point de contrainte V. Chaque contrainte dispose d'une ou plusieurs fonctions d'extrusions, selon le type de celles-ci. On peut par exemple, dans un espace de dimension 3, avoir trois fonctions d'extrusions $f : R \rightarrow R$, ou une seule fonction $f : R^3 \rightarrow R$, etc. Dogme^{RV} ne permet que l'utilisation de fonctions dont le profil est $f : R \rightarrow R$, mais dans le cas général on peut appliquer d'autres fonctions qui permettent par exemple d'optimiser des critères supplémentaires, comme le volume.

Dogme^{RV} propose deux types de fonctions d'extrusion : les fonctions paramétrables, que l'utilisateur peut ajuster à l'aide d'un ou plusieurs paramètres et qui ont une formulation mathématique explicite, et les fonctions libres, qu'il peut dessiner et ajuster à main levée.

Fonctions paramétrables

Dogme^{RV} dispose de trois fonctions paramétrables (eq. 3.2 et 3.3), dont deux sont ajustables par un paramètre. Intuitivement, ce paramètre contrôle la pente de la transition entre la zone déformée et la zone non déformée. La manipulation du paramètre fait appel au contrôle d'application et notamment à un composant *potentiomètre linéaire*. La figure 3.7 illustre cette interface ainsi que les effets du paramètre sur la déformation.

$$I(x) = \begin{cases} f_i(x)^n & \text{si } n < 1 \\ f_i(x^{1/n}) & \text{si } n > 1 \end{cases} \tag{3.2}$$

$$\begin{array}{ll} \text{Fonction porte :} & f_i(x) = 1 \\ \text{Fonction } G_0\text{-continue :} & f_i(x) = 2x^3 - 3x^2 + 1 \\ \text{Fonction } G_0\text{-discontinue :} & f_i(x) = 1 - x \end{array} \tag{3.3}$$

Fonctions libres

L'utilisateur peut également dessiner librement la forme de la fonction d'extrusion. Étant donné le type de fonctions supportées par le prototype, cette fonctionnalité requiert la capacité à tracer une courbe $2D$, comme on dessinerait une courbe $f(x) : R \rightarrow R$ sur une feuille de papier.

C'est précisément cette métaphore qui est proposée au travers du composant *Stratch Board*[1]. Une zone plane 2D capture les intersections avec le rayon laser, qui *dessine* ainsi une trace derrière lui. Les différentes positions sont échantillonnées régulièrement, puis traduites en fonction $f(x) : [0..1] \rightarrow [0..1]$.

[1] feuille de croquis

Il est souvent difficile de dessiner à main levée une forme parfaitement lisse (ce qui est en général souhaité pour une fonction d'extrusion, puisqu'une fonction lisse donnera lieu à une déformation lisse), d'autant que l'imprécision et les tremblements sont démultipliés par la distance entre la main et le plan de dessin. Pour corriger ce défaut, nous considérons l'intersection du rayon avec le plan de dessin non plus comme un point infiniment petit, mais comme un disque de rayon fini, centré sur le point d'intersection du rayon avec le plan. A un instant donné, lors du dessin, les échantillons e_i de la courbe E sont modifiés de la manière suivante pour former courbe modifiée E' :

$$E' = \{e_i \in E, e_i * \alpha(i - l_x * card(E)) * l_y\} \tag{3.4}$$

$$\alpha(d) = \begin{cases} 1 - \frac{d}{R} & \text{si } d < R \\ 0 & \text{sinon} \end{cases} \tag{3.5}$$

avec l_x et l_y l'abcisse et l'ordonnée de l'intersection du rayon avec le plan (entre 0 et 1), α la fonction qui définit l'interpolation entre le tracé du laser et la courbe existance, sur un rayon R.

Ainsi, à chaque passage, plusieurs points échantillons sont modifiés de part et d'autre de l'intersection avec le rayon, selon une interpolation qui garantit un raccord fluide avec la courbe existante. Le dessin de la courbe en plusieurs fois est également simplifié puisque les raccords sont automatiquement lissés entre plusieurs tracés. La Figure 3.8 page 68 illustre schématiquement le principe, qui est similaire aux pinceaux à bords adoucis de nombreux logiciels de dessin du commerce.

Outre la saisie directe, l'utilisateur peut manipuler la courbe à l'aide de tangentes et points de passage. Comme pour les chemins de déformation, la fonction d'extrusion est transformée en courbe de Bézier cubique par morceaux, sur lesquels on peut modifier les points. Pour être fournie au moteur de déformation, la courbe est ensuite échantillonnée régulièrement selon x. Il est à noter que la courbe ainsi que les tangentes restent à tout moment contraintes dans un plan. Ces techniques sont illustrées par la figure 3.9 (voir également la figure 3.4 pour un gros-plan sur la palette de propriétés).

3.4.5 Volume de voxels

Les volumes de voxels forment un groupe de volumes d'influence particulier. En effet, ils combinent en une seule entité à la fois le volume d'influence et la fonction d'extrusion. Les voxels peuvent ici prendre des valeurs numériques, comprises entre 0 et 1. Leur valeur nous renseigne sur la «quantité» de déformation que la contrainte communique au point considéré, ce qui remplace intégralement la fonction d'extrusion. En fait, le volume de voxels est une fonction d'extrusion ne dépendant pas de la distance euclidienne entre le point déformé et le point de contrainte, mais dépendant du vecteur entre les deux points. Le volume de voxels échantillonne donc une fonction d'extrusion quelconque dont on se resert durant la

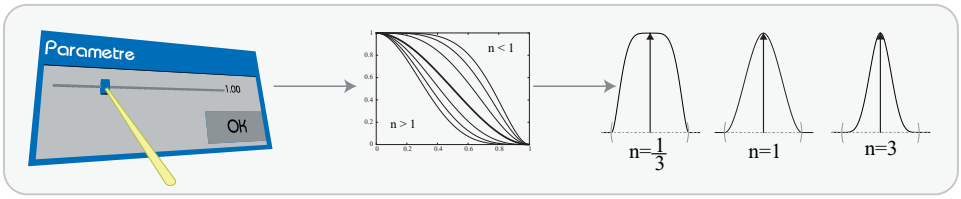


FIG. 3.7 – Réglage du paramètre et effets sur la déformation.

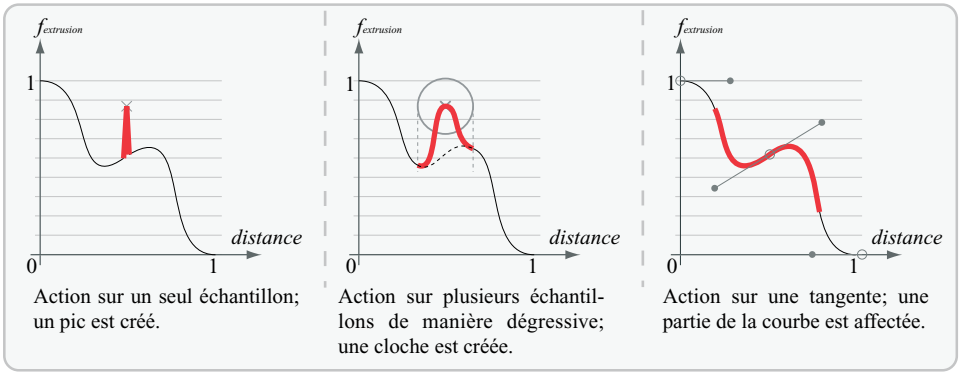
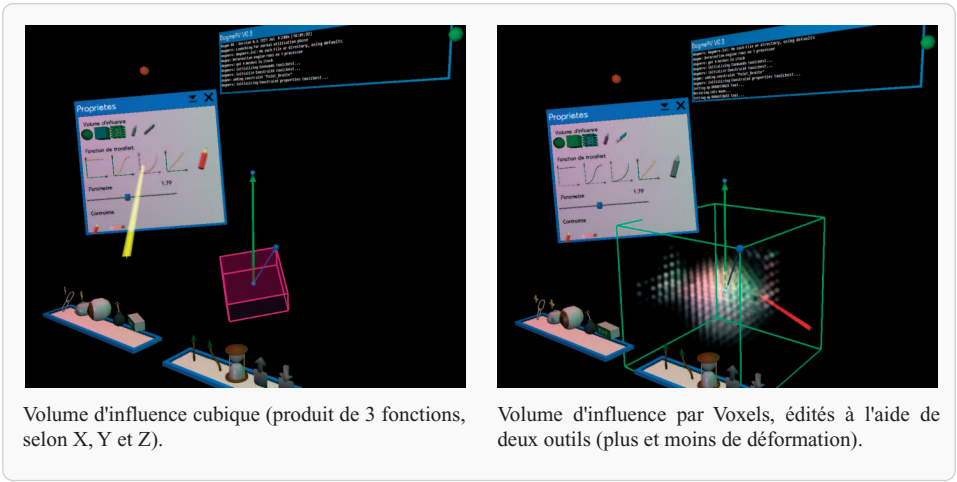


FIG. 3.8 – Principe de la saisie d'une fonction d'extrusion. En rouge gras, la portion de courbe affectée par la manipulation.



Volume d'influence cubique (produit de 3 fonctions, selon X, Y et Z).

Volume d'influence par Voxels, édités à l'aide de deux outils (plus et moins de déformation).

FIG. 3.9 – Interface de saisie du volume d'influence.

déformation. Le volume dans lequel sont stockés les voxels est centré sur le point de contrainte V. Son orientation et sa taille se manipulent de manière identique aux volumes d'influence classiques. Il est à noter que le changement de taille du volume ne change pas le nombre de voxels qu'il contient.

Le volume de voxels peut également être pré-initialisé à partir de l'intérieur d'un objet triangulé quelconque, ou d'une fonction d'extrusion implicite ou explicite. Une fois créé, le contenu du volume peut être édité directement par l'utilisateur au moyen de deux outils, dont le comportement est similaire à un aérographe[2]. L'un des outils va «déposer de la déformation» (c'est à dire augmenter la valeur des voxels rencontrés), tandis que l'autre va en retirer (en diminuant la valeur des voxels rencontrés). En déplaçant l'outil dans le volume, on peut ainsi dessiner la forme de la déformation autour de la contrainte, en volume. Les outils sont de forme sphérique de rayon 3 voxels. L'intensité de la déformation (ou de l'absence de déformation) est dégressive sur les bords de l'outil, afin de favoriser des tracés –et donc des déformations– lisses.

Afin de mettre cet outil en œuvre, il faut également disposer d'une représentation correcte des voxels eux même. L'utilisation et la manipulation que nous faisons des voxels requièrent une visualisation particulière, qui montre tout l'intérieur du volume et non une isosurface comme c'est fréquemment le cas. Nous avons opté pour un rendu par *splatting*, ou projection. Celui-ci consiste à afficher, au centre de chaque voxel, une tache de couleur dont l'intensité est fonction de la valeur du voxel. En jouant sur la transparence et la couleur de chaque tache, on peut transmettre une certaine notion de volume. Les taches sont affichées de manière à toujours faire face à l'utilisateur, ce qui suggère que ces taches sont sphériques. La couleur des taches est calculée en fonction des gradients du voxels et des sources lumineuses. Par conséquent, à l'intérieur d'un volume dont les valeurs sont presque constantes, les gradients sont quasiment nuls. La couleur résultante devient donc noire, et la tache invisible. En réalité, seules les zones dont les gradients sont élevés sont réellement visibles, car leur couleur est suffisamment claire pour être perceptible. On peut bien entendu passer les gradients dans une fonction de transfert (type correction gamma par exemple), en prenant garde à ne pas surexposer l'affichage.

Pendant la manipulation, une seconde aide visuelle vient s'ajouter à l'affichage : la position courante de l'outil, lorsqu'il se trouve à l'intérieur du cube de voxels, est repérée par une croix, centrée sur l'outil et partant des bords du cube. Ceci permet d'avoir de mieux se rendre compte de la position de l'outil à l'intérieur du volume. En effet, comme l'ont souligné Zhai et al. [ZBM94], un affichage naïf des objets translucides –comme le sont les voxels– ne communique pas beaucoup

[2] L'aérographe est un outil de dessinateur qui permet de projeter de microscopiques gouttes de peinture sur une surface. L'artiste en contrôle le débit, et peut ainsi créer des dégradés très réguliers. Son fonctionnement est apparenté à une bombe de peinture.

d'informations de profondeur, or cette information est importante lorsque l'on souhaite modifier un endroit particulier du volume.

3.5 Contrôle d'application

Le contrôle d'une l'application représente une partie omniprésente de celle ci. Beaucoup d'opérations reposent sur celui-ci, au moins indirectement, comme créer une contrainte, changer la forme de son volume d'influence ou modifier le type d'affichage. Un accent tout particulier a été mis sur ce dernier, de manière à pouvoir en changer pour expérimenter diverses présentations et métaphores.

Il apparaît à l'usage que le contrôle d'une d'application a une importance de plus en plus importante au fur et à mesur que la taille et la complexité de l'application augmente, en termes de nombre d'opérations et de modes possibles. S'il est tout à fait possible de se satisfaire d'une interface rudimentaire voire inexistante lorsque la seule action possible est de déplacer et tourner une théière virtuelle, il en va tout autrement lorsque les possibilités d'interactions augmentent. Il est d'une part nécessaire de permettre à l'utilisateur de les activer, mais il est également préférable de lui permettre de le faire aisément et efficacement. Le rôle du contrôle d'application est d'une part d'offrir les possibilités, mais aussi et surtout de les offrir de manière efficace et sensée. Comme nous n'avions pas d'idée précise *à priori* sur le type d'interfaces qui fonctionneraient le mieux, nous nous sommes laissées le champ libre pour une certaine expérimentation –par l'intermédiaire notamment de la librairie *libSelect*, dont nous reparlerons au chapitre 5. Nous avons par conséquent le choix entre plusieurs types de contrôle, que nous allons décrire dans les paragraphes suivants, avec leurs avantages et inconvénients respectifs. Ces interfaces sont illustrées Figure 3.10 page 71. Elles ont été testées auprès d'utilisateurs. Le test ainsi que son résultat sont présentés en section 3.6, en page 74.

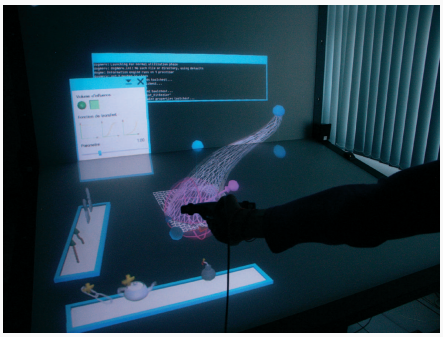
3.5.1 Agencement des commandes

La hiérarchie des commandes du programme est en grande partie commune à toutes les interfaces de contrôle, sauf pour les fenêtres 3D et palettes d'outil puisqu'elles supportent de manière transparente la présence de plus d'une fenêtre à l'écran. La figure 3.11 illustre la hiérarchie des commandes de Dogme^{RV}.

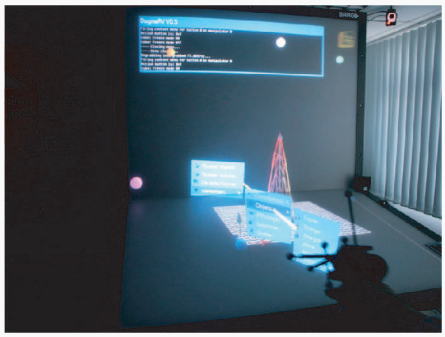
On peut scinder les commandes en deux grandes familles : celles dont on dispose en permanence, et celles relatives à une contrainte dont on ne dispose que lorsqu'une contrainte est active. On peut ici faire le parallèle entre la barre de menus d'une application, toujours présente, et une feuille de propriétés ou un menu contextuel.

3.5.2 Fenêtres 3D

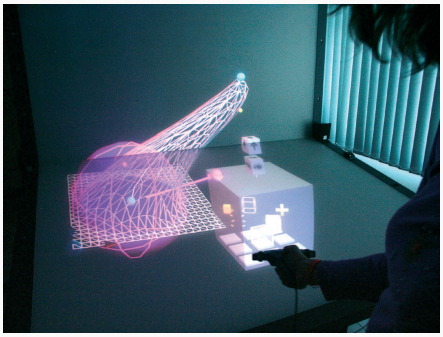
La première interface que nous avons implémentée fût copiée de près de l'environnement graphique habituel de nos ordinateurs de bureau. Les différentes options



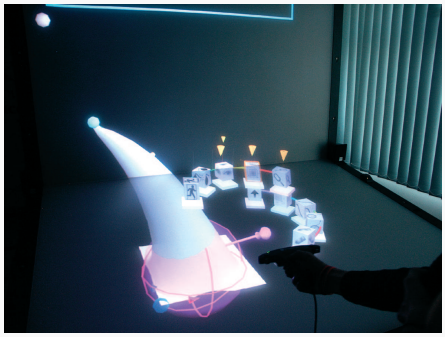
WIMP
Sélection des commandes par rayon laser virtuel.



Menus 2D convertis
Sélection par rayon



C^3
Sélection des commandes mouvement dans une direction (droite, haut, arrière, etc.)



Spin Menu
Sélection des commandes rotation du poignet.

FIG. 3.10 – Différentes interfaces pour la même application.

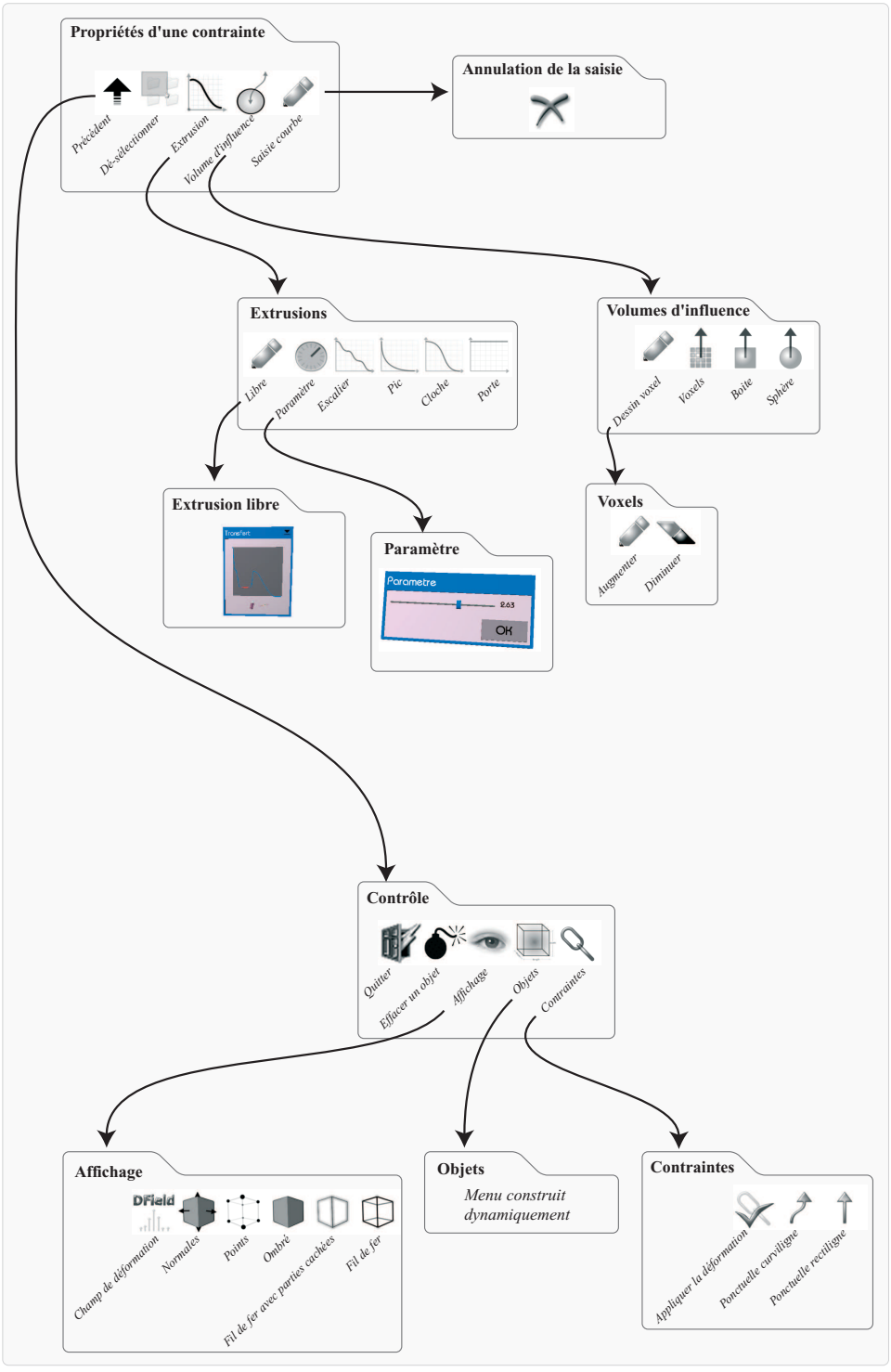


FIG. 3.11 – Hiérarchisation des commandes dans Dogme^{RV}

sont présentées dans une ou plusieurs fenêtres conteneur, dans lesquelles sont rangés des boutons d'action, des cases à cocher ou des boutons radio principalement. La sélection, au lieu de se faire à la souris, se fait à l'aide du rayon laser virtuel, là aussi extension presque directe du pointeur de souris à la 3D. Il s'agit donc là d'une classique interface $2D\frac{1}{2}$.

Malheureusement, même si une telle interface est hautement compréhensible grâce au parallèle très fort avec les interfaces usuelles, elle souffre d'un handicap majeur : l'utilisateur doit, dès qu'il veut valider un bouton, interrompre ce qu'il est en train de faire. Dans certains cas, ceci n'est pas un grand problème, notamment si le bouton à activer se trouve à proximité de l'endroit d'intérêt. En revanche, si la tâche qu'il accomplissait demandait une manipulation précise, ou que le bouton à actionner se trouve éloigné de la zone d'intérêt, devoir l'abandonner –et y revenir plus tard– pose un problème de continuité dans le flux cognitif de l'utilisateur. Ce problème existe également en 2D, mais est plus accentué en 3D de par le plus grand nombre de degrés de libertés du positionnement. Nous avons par conséquent cherché à minimiser autant que possible la *cassure* causée par le contrôle d'application dans la manipulation de l'utilisateur.

Puisque les fenêtres 3D sont partiellement insatisfaisantes, nous les avons remplacées par des palettes d'outils. Il s'agit de fenêtres 3D, dont les boutons (ou icônes) sont comme posés au dessus du conteneur (Figure 3.12). Celles-ci peuvent tirer profit de la disposition des écrans d'un workbench si on les affiche sur l'écran inférieur, a proximité de l'utilisateur, d'autant que cet espace est rarement utilisé pour les données de l'application. Les palettes prennent nettement moins de place, mais présentent cependant les mêmes limitations que les fenêtres 3D (nécessité de quitter la manipulation pour «cliquer» sur une icône).

3.5.3 Menus déroulants

Les menus déroulants constituent une partie essentielle de toute interface graphique moderne. Ils permettent, en un espace très restreint, de placer un nombre important de commandes, tout en les organisant de façon logique en menus et sous-menus. Comme nous l'avons déjà évoqué, ils ont été adaptés à un environnement tridimensionnel très tôt, en ce qu'il convient d'appeler les *menus 2D convertis* [JE92].

Nous disposons également d'une interface basée sur cette métaphore. Leur contrôle peut se faire soit à l'aide d'un rayon virtuel. Notons que nous avons choisi d'utiliser exclusivement des menus contextuels afin de libérer autant d'espace visuel que possible sur les écrans pour les tâches de manipulation. Les menus s'ouvrent orthogonalement à la direction du regard de l'utilisateur, dans le prolongement de sa main. Les sous-menus s'ouvrent en angle par rapport à leur menu parent afin d'en faciliter la lecture.

3.5.4 Command and Control Cube

Dans la veine des menus, mais vus sous un angle totalement différent, on peut noter l'existence du Command and Control Cube, ou C^3 en abrégé. Comme nous l'avons déjà évoqué, cette métaphore se base sur le postulat qu'il est plus sûr et plus rapide d'effectuer un mouvement dans une direction donnée qu'un mouvement d'une amplitude précise.

Chaque action du menu est accessible selon une direction différente. En trois dimensions, puisque l'on peut aller dans un sens ou dans l'autre selon chaque dimension, cela nous donne un total de 26 combinaisons différentes, plus une combinaison spéciale qui est l'absence de direction, agencées sur un cube de 3x3x3 cases. Par définition, l'absence de mouvement indique également l'absence de sélection. Les 26 autres peuvent chacune accueillir une commande, que l'utilisateur peut actionner en effectuant un mouvement dans sa direction à partir du centre. Les auteurs proposent une représentation des actions comme des petits cubes texturés. Nous proposons d'utiliser des icônes 3D et un petit texte descriptif au dessus de l'élément sélectionné. Ceci dans le but de permettre un apprentissage plus aisé des commandes pour les utilisateurs novices.

3.5.5 Spin Menu

Le Spin Menu, qui fera l'objet du chapitre suivant, est une alternative intéressante aux menus déroulants. Il propose de disposer les éléments non pas linéairement mais sur un arc de cercle, et d'y adjoindre une manipulation de rotation spécifique. On peut ainsi proposer une manipulation totalement adaptée à la métaphore –ce qui n'est pas le cas des menus 2D convertis. Dans notre implantation, la manipulation consiste à tourner le poignet qui tient le périphérique d'interaction dans le plan horizontal pour bouger l'élément sélectionné.

Cette métaphore apparaît après expérimentation et adaptation comme une alternative efficace aux imposantes fenêtres 3D et aux menus déroulants, permettant une manipulation rapide et aisée.

3.6 Comparaison des interfaces de contrôle

Le prototype étant doté de plusieurs interfaces de contrôle, il nous a semblé intéressant de tenter une comparaison entre celles-ci. Nous avons cherché à mettre en évidence la présence ou l'absence d'apprentissage (gain de temps du à la répétition d'une même manipulation) sur les différentes métaphores. Nous avons également souhaité recueillir des avis subjectif de personnes ne faisant pas nécessairement partie du domaine de l'informatique graphique et de la réalité virtuelle, afin d'évaluer les métaphores selon quelques critères présentés plus bas.

Pour cela, nous avons fait appel à des volontaires, devant effectuer une manipulation prédéfinie, sous deux types d'interfaces différents. Pour chaque utilisateur, nous avons procédé de la manière suivante :

- Explication rapide du prototype et de son fonctionnement.
- Quelques minutes de familiarisation de l'utilisateur avec l'environnement virtuel.
- Explication informelle de la tâche à accomplir.
- Réalisation de la tâche par l'utilisateur, avec la métaphore du *rayon laser* et des *fenêtres 3D*. Les différentes instructions étaient données oralement au fur et à mesure de l'avancement par des commentaires pré-enregistrés, ainsi que par un affichage textuel.
- Réalisation de la tâche par l'utilisateur une seconde fois, en utilisant soit l'interface C^3, soit l'interface *Spin Menu*. Les menus déroulants n'ont pas été testés.
- Discussion à chaud sur les impressions suite aux deux manipulations, et remplissage d'un questionnaire (Figure B.1 page 161).

La tâche a réaliser demandait entre 5 et 20 minutes, et consistait en la construction d'une scène 3D composée d'une table (une boîte déformée par 4 contraintes) et d'un cactus (un cylindre déformé par 2 contraintes). Un exemple de scène effectuée est montrée Figure 3.13 page 76. Il n'était pas demandé de faire particulièrement attention à l'esthétique de la scène. Le détail des opérations est fourni en annexe (Figure B.4 page 164). Durant la manipulation, nous enregistrions le temps mis à passer d'une étape à l'autre et le nombre d'erreurs commises. Ceci a été répété pour les 19 volontaires. Le tableau 4.1 donne de plus amples informations à leur sujet. Pour ce test, les utilisateurs experts sont des personnes ayant déja cotoyé le plan de travail virtuel à l'occasion de développements notemment. Les utilisateurs novices découvraient totalement l'environnement. Notons enfin que le nombre de commandes était le même pour les trois types de contrôle, les fenêtres étant moins hiérarchisées que les menus puisqu'elles permettent un plus grand nombre d'icônes sur un même conteneur. La taille d'un menu particulier dans la hiérarchie ne dépassait en outre pas les 7 éléments, la profondeur maximale de cette hiérarchie étant de 3 (les menus *Extrusion libre* et *Voxels* ayant été supprimés afin d'éviter des erreurs de manipulations).

Après analyse des résultats, nous espérions pouvoir déceler plusieurs indicateurs à partir des temps enregistrés. Malheureusement, nous nous sommes aperçues à regret que le prototype était trop complexe à appréhender pour des personnes non habituées à la modélisation géométriques. En effet, bon nombre d'utilisateurs ont avoué se sentir perdus dans la multitude de termes spécifiques utilisés dans les instructions orales (contrainte, volume d'influence, etc.). Nous avions par exemple espéré détecter une amélioration du temps mis pour créer le dernier pied de la table par rapport au premier (les quatre étant faits exactement de la même manière). Malheureusement, les écarts d'une personne à l'autre et d'un pied à l'autre sont trop importants. De plus, certains utilisateurs accordaient d'avantage d'importance à l'esthétique de la scène finale, et donc passaient plus de temps à placer précisément les contraintes et objets que d'autres. Peut être la manipulation était-elle trop lâchement guidée ? Il serait en tous cas faux de tirer une quelconque conclusion à partir des temps de manipulation.

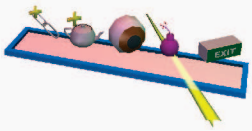


FIG. 3.12 – Palette d'outils 3D.

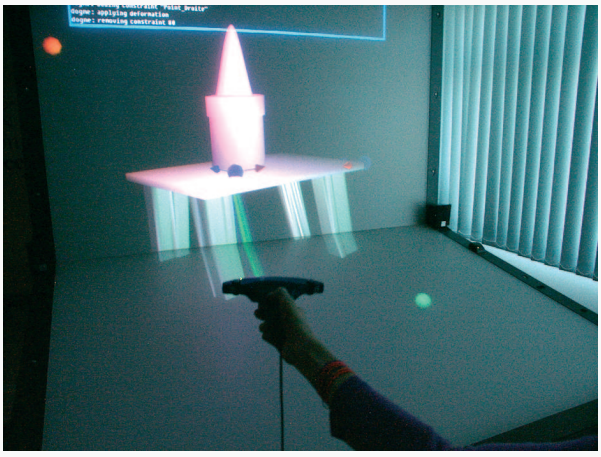


Table & Cactus

Les utilisateurs avaient pour tâche de contstruire une table à partir d'un plan et un cactus à partir d'un cylindre en y apposant les contraintes nécessaires (manipulation très guidée, d'environ 20 minutes)

FIG. 3.13 – Exemple de scène construite par les utilisateurs

Il ne reste alors que les commentaires et notes subjectives assignées par chaque utilisateur aux différents critères mis en avant par le questionnaire. La Figure 3.14 page 78 montre la moyenne de ces différentes notes. Les explications sont détaillées ci dessous. Celles-ci montrent que l'interface préférée de nos utilisateur restent les fenêtres 3D. On peut attribuer ceci à l'habitude de ce type d'interfaces. Le *Spin Menu* arrive en seconde place, apprécié pour son esthétique et la facilité de mémorisation de la position des éléments, comme nous le verrons dans le chapitre suivant. Le C^3 arrive en dernière position, et semble, aux dires des utilisateurs, compliqué à utiliser car l'organisation en 3D des commandes est inhabituelle et leur demande un effort d'adaptation plus important. Pourtant, les menus de notre applications tenaient tous sur l'étage central du cube, dont la manipulation se réduisait de fait à deux dimensions. Il est important de noter que la métaphore n'est pas utilisée de façon optimale comme définie par les auteurs. Ceux-ci préconisent la métaphore pour une manipulation *en aveugle* par des utilisateurs experts, qui connaissent l'agencement du menu par coeur et n'ont par conséquent pas besoin de retour visuel. Nos utilisateurs n'étaient malheureusement pas des experts, et se servaient abondamment du retour visuel pour guider leur manipulation. De plus, il faut signaler que notre équipement de suivi de mouvement souffrait de quelques problèmes de précision en position, et introduisait parfois un bruit pouvant aller jusqu'à 5*cm* (mesurés) autour de la position du périphérique. Ceci influe particulièrement sur les métaphores de fenêtres 3D et du C^3 qui requierent une certaine précision en positionnement. On peut probablement imputer une partie de la faible note du C^3 selon le critère d'aisance de manipulation à ce bruit. Cependant, la métaphore du rayon laser souffrait du même problème mais reçoit une note très élevée, ce qui permet de croire que ce bruit seul n'explique pas l'écart des scores, mais incite à en relativiser la portée tout de même.

Les critères selon lesquels les utilisateurs devaient noter les métaphores, entre 1 (mauvais) et 5 (excellent) sont les suivants :

Appréciation générale L'impression générale que vous a procuré la technique dans son ensemble : avez-vous aimé ou non.

Facilité de localisation des éléments Comment jugez-vous la clarté de la représentation graphique ? On ne peut pas localiser l'élément que l'on cherche ? On y voit parfaitement clair ?

Intuitivité de la manipulation Est-il facile de comprendre comment utiliser la technique ?

Aisance de la manipulation Est-il facile de piloter la technique ?

Effet mémoire Est-ce que vous avez l'impression que l'habitude peut vous aider à utiliser mieux la technique ?

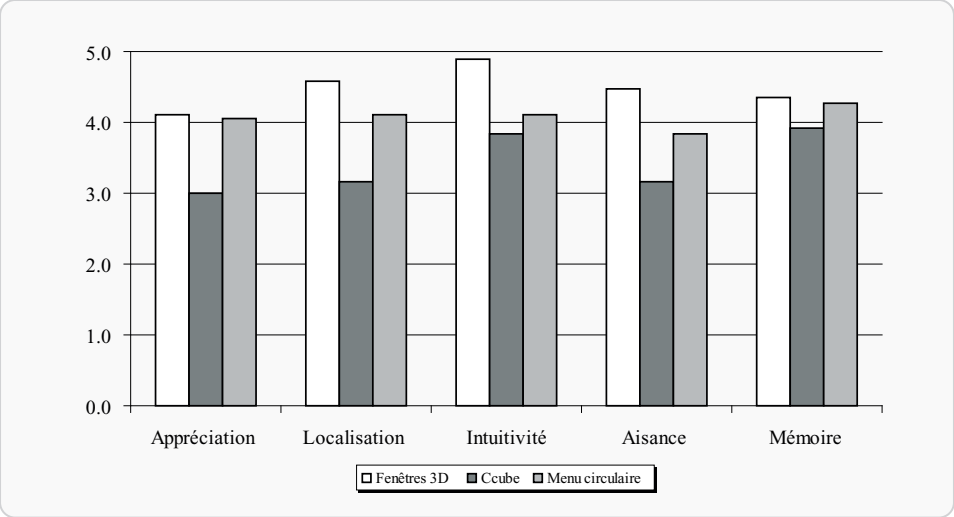


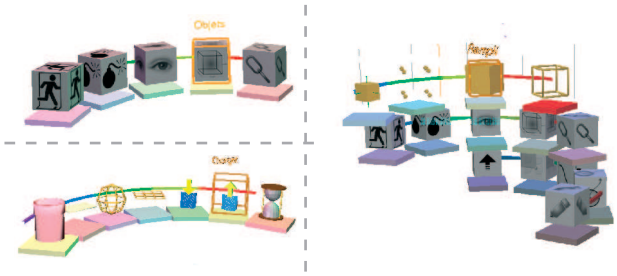
FIG. 3.14 – Évaluation des différentes interfaces par les utilisateurs

3.7 Conclusion

Comme nous venons de le voir, Dogme^{RV} est une application relativement complexe, et offre de multiples interactions. Nous nous sommes particulièrement intéressées au contrôle de cette application. Après avoir fait passer des utilisateurs en test sur ce logiciel, il semble que ceux-ci préfèrent à l'unanimité les fenêtres 3D. Cependant, celles-ci continuent d'être encombrantes et d'obliger à interrompre fréquemment sa tâche pour aller «cliquer» sur un bouton. On peut supposer qu'une telle interface est particulièrement adaptée aux utilisateurs débutants, mais handicape la performance d'un expert. Au contraire, le C^3 est plutôt réservé aux experts, connaissant déjà parfaitement l'application et ne cherchant qu'à être efficace dans leur tâche. Le Spin Menu se situe alors entre ces deux extrêmes, en ne handicapant les experts qu'un minimum, tout en permettant aux débutants de se repérer facilement.

Chapitre 4

Le Spin Menu



4.1 Introduction

Nous allons dans ce chapitre proposer une métaphore de menu conçue pour les environnements virtuels, pouvant être manipulée aisément et rapidement par des utilisateurs experts ou non. Il s'agit d'une métaphore appartenant à la famille des menus circulaires, que nous proposons de baptiser le *Spin Menu*. Celle-ci permet de créer une hiérarchie de commandes, analogue à n'importe quel menu déroulant classique, représentés sur un arc de cercle et propose une manipulation adaptée à cette disposition. La disposition circulaire des commandes d'un menu n'est pas en soi une idée nouvelle. En effet, elle a été introduite en réalité virtuelle en 1994 par Liang dans une application de modélisation géométrique, et fait régulièrement parler d'elle en 2D[Hop04]. Cette métaphore avait été proposée car elle correspondait au matériel dont l'auteur disposait alors : un périphérique sensible à la rotation sur un axe. Dans ces conditions, quoi de plus normal que d'afficher les éléments du menu sur un cercle ? Plus récemment, Weshe et al. [WD00] ont montré un menu de forme arrondie rappelant fortement la famille des menus circulaires, mais dont la manipulation se fait par un rayon, rapprochant ainsi davantage leur technique des menus 2D convertis que des menus circulaires modulo sa représentation légèrement incurvée. Même si les menus arrondis ou circulaires resurgissent régulière-

ment en environnements virtuels, la métaphore reste peu utilisée, et surtout aucune étude poussée n'a été faite à son sujet.

Nous nous proposons, à partir des idées de menu circulaire et de manipulation par rotation construire une métaphore de menu complète et la confronter aux utilisateurs afin d'en valider les différents aspects. Nous décrirons dans un premier temps la métaphore de Liang, sur laquelle nous basons en partie nos idées. Nous décrirons également le menu de Weshe. Nous verrons ensuite les apports que nous y amenons. Dans un troisième temps, nous verrons l'ajustement des différents paramètres de la métaphore, appuyés par des tests utilisateurs. Enfin, nous verrons le passage à la hiérarchie, avec les problèmes posés et les solutions proposées.

4.2 La métaphore de Liang

Le menu circulaire introduit par Liang consistait en un anneau sur lequel étaient disposées des icônes 3D représentant les différentes commandes que l'utilisateur pouvait activer. Un trou dans l'anneau indiquait l'élément sélectionné, c'est-à-dire celui qui serait activé. Liang montrait comme exemple un menu permettant de créer des primitives géométriques simples comme un cube, une sphère ou un tore. La Figure 4.1 page 83 illustre cette représentation. La manipulation s'effectuait à l'aide d'un périphérique appelé *the bat*, aujourd'hui disparu. Il permettait de détecter la rotation autour d'un axe à la manière d'un potentiomètre. C'est ainsi que l'utilisateur tournait le menu et ses éléments, pour amener l'élément d'intérêt dans le trou formé par l'anneau pour ensuite l'activer. Cette technique, totalement fonctionnelle, pose tout de même le problème du faible nombre d'éléments que le menu peut accueillir, ce qui explique peut-être au moins en partie que l'on n'ait vu que très peu d'utilisations de cette métaphore sous cette forme depuis.

Parmi les éléments jouant en la défaveur de cette métaphore, on peut noter que d'une part les actions sont représentées comme des icônes 3D, ce qui implique que toute action puisse être représentée par un symbole compréhensible. On imagine assez aisément que ce n'est pas le cas de toutes les commandes, comme par exemple *modifier le vecteur de nœuds d'une courbe NURBS*. D'autre part, la représentation se faisant autour d'un cercle, faisant ainsi 360°, on arrive rapidement à un nombre maximal d'éléments au dessus duquel la précision devient insuffisante et le menu trop chargé. Bien entendu, nous savons qu'il est fortement déconseillé de concevoir des menus comptant plus d'une dizaine d'éléments, pour des raisons de mémorisation et d'apprentissage [Sch83], mais aucune solution n'était proposée par les auteurs pour pallier à ce problème si cela s'avérait nécessaire.

Néanmoins, la disposition circulaire des éléments et la manipulation basée sur la rotation nous semble intéressante, aussi avons-nous décidé de l'utiliser et de l'étendre.

4.3 Intérêt des menus circulaires

4.3.1 Représentation et mémorisation

La représentation graphique que propose cette métaphore est facile à appréhender même sans entraînement particulier : les éléments sont disposés le long d'un cercle –éventuellement d'une portion de cercle, de sorte que chaque élément ait au plus deux voisins (un «à droite» et un «à gauche»). On se retrouve finalement avec une configuration analogue aux menus déroulants classiques, ce qui a un aspect familier pour les utilisateurs découvrant la métaphore et en facilite la prise en main. De plus, la disposition géométrique des éléments lorsque le menu apparaît favorise une mémorisation dichotomique par rapport à l'élément central, que les utilisateurs ont naturellement tendance à utiliser. Ainsi, lorsque l'on demandera d'activer l'élément *Créer objet*, se trouvant visuellement à droite de l'élément central, il ne sera pas rare de recevoir comme commentaire oral lors des tests une indication comme «c'est celui qui est à un peu à droite».

4.3.2 Manipulation et main non dominante

La manipulation demandée par la métaphore, à savoir *tourner le poignet* dans le plan horizontal, est peu coûteuse en concentration et est peu sensible aux *tremblements* involontaires induits par le maintien d'une position dans l'espace. De plus, puisque aucun déplacement de la main n'est requis, l'utilisateur reste, durant le temps de la manipulation du menu, dans la zone même ou il manipulait au préalable. A titre de comparaison, une métaphore telle que le laser virtuel associé à des fenêtres 3D demande fréquemment d'interrompre sa tâche pour déplacer le laser sur la fenêtre contenant les commandes utiles, puis de revenir à ce que l'on faisait.

Ces aspects nous ont suggéré que la métaphore devait pouvoir être manipulée par la main non dominante. Ceci permet de dissocier le contrôle d'application et la manipulation des données sur les deux mains, renforçant ainsi le côté non dérangeant de la métaphore. Une métaphore telle que le rayon laser virtuel par exemple est très difficile à manipuler à la main non dominante, puisqu'il requiert une grande précision en position. La manipulation du Spin Menu est suffisamment simple et relativement peu demandeuse de précision dans l'absolu pour que la main non dominante s'avère capable d'effectuer une manipulation tout à fait satisfaisante. Ceci sera détaillé et étayé par des expériences dans la section 4.4.7

4.3.3 Quelques notions

Avant d'aller plus avant, nous allons définir quelques termes afin de simplifier les discussions suivantes.

Anneau

Le Spin Menu est composé d'un ou plusieurs *anneaux*. Un anneau est défini comme un arc de cercle, contenant une ou plusieurs commandes définissant un menu. La métaphore elle-même peut être composée de menus et de sous-menus –anneaux et sous-anneaux dans la terminologie du Spin Menu. Bien sur il s'agit là d'une vue de l'esprit, puisqu'il n'y a aucune différence en terme de manipulation ou de visualisation entre un anneau et un autre –à part bien sur les icônes des commandes et leurs actions. Notons également qu'une portion d'anneau n'est pas nécessairement affichée à l'écran, ce choix étant principalement esthétique. La présence visuelle d'une portion d'anneau peut toutefois s'avérer utile si les éléments du menu sont très espacés.

Orientation de référence

Lorsque l'utilisateur fait apparaître le menu, l'orientation de son poignet est mémorisée. Cette orientation sert de référentiel pour toutes les mesures d'angles suivantes. Nous appellerons cette orientation initial l'*orientation neutre*. L'affichage initial du menu est fait de sorte à ce qu'il soit centré sur l'élément du milieu. L'orientation neutre correspond toujours à l'affichage centré du menu (figure 4.3).

Angle physique et logique

La manipulation de la métaphore fait intervenir deux notions d'angles : les angles *physique* et *logique* (Figure 4.4). Les angles physiques mesurent les rotations entre la direction indiquée par le poignet (ou plus précisément le matériel de suivi de mouvement), et l'orientation de référence. Les angles logiques mesurent les rotations visuelles de la métaphore, entre son affichage de référence, et son affichage courant. De plus, puisque l'élément sélectionné est repéré visuellement, les angles logiques servent également à la détermination de l'élément actif.

Annulation du menu

Dans toute métaphore modale, il faut prévoir une option de sortie sans aucune validation. Dans les menus déroulants 2D conventionnels, il suffit de cliquer ailleurs sur l'écran que dans la zone occupée par le menu, ou d'appuyer sur la touche *Echap*. En réalité virtuelle, nous ne disposons d'aucun clavier, appuyer sur une touche n'est donc pas envisageable. De même, «cliquer en dehors du menu» ne convient pas, puisque durant la manipulation du menu, nous ne disposons pas directement d'une notion de position. Une autre possibilité a été retenue, celle d'un geste spécifique. S'il faut effectuer des rotations dans le plan horizontal pour déplacer la sélection, nous n'avons qu'à utiliser le plan orthogonal pour notre geste de fermeture. Ainsi, pour annuler l'ouverture du menu sans effectuer aucune action, l'utilisateur devra effectuer une rotation vers le haut ou le bas suffisamment ample –pour ne pas fermer le menu par erreur. Nous avons choisi une amplitude de 45°,

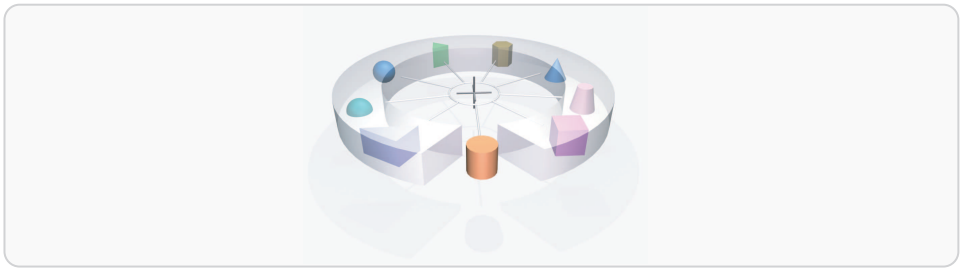


FIG. 4.1 – Le menu circulaire introduit par Liang (figure d'après [LG94]).

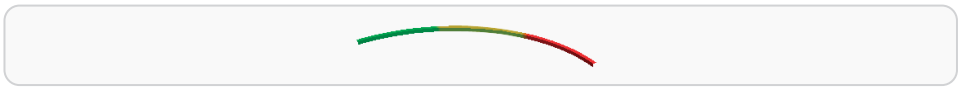


FIG. 4.2 – L'anneau d'un Spin Menu

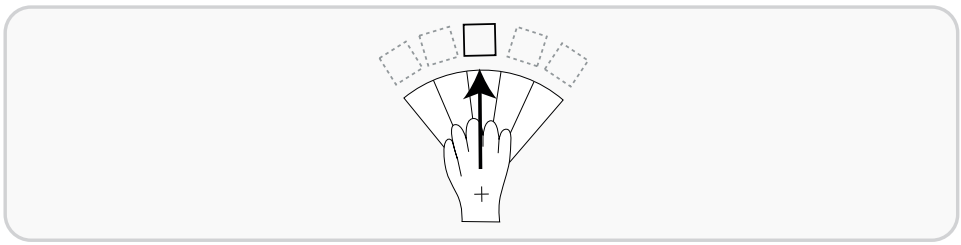


FIG. 4.3 – Orientation de référence

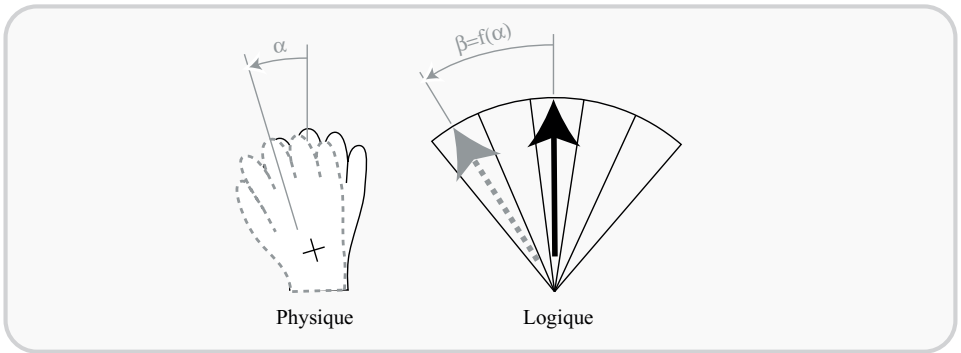


FIG. 4.4 – Angles physiques et logiques

qui convient bien et ne provoque jamais de fermeture par erreur. Une illustration schématique du geste de fermeture est donnée par la figure 4.5.

4.3.4 Apports du Spin Menu

Bien que basé sur les travaux de Liang, le Spin Menu que nous proposons ici prend quelques distances et propose quelques modifications importantes.

Manipulation

La manipulation que nous proposons n'est pas totalement identique à celle évoquée par Liang. En effet, alors qu'il proposait de tourner le poignet sur lui-même (dans l'axe de l'avant-bras), nous proposons une rotation autour de l'axe normal à la paume de la main (axe vertical). La Figure 4.5 illustre cette manipulation. Ce choix provient en partie du périphérique que nous utilisons, qui favorise une telle rotation vue la façon dont il est pris en main. Il serait a priori envisageable d'adapter la manipulation à un autre type de périphérique, mais ceci demanderait d'optimiser bon nombre de paramètres pour obtenir à nouveau une manipulation optimale. Le principe de manipulation est décrit dans l'automate à état de la page 85.

Affichage

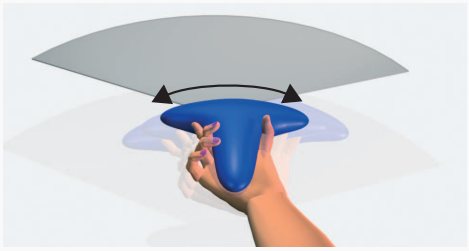
Le menu n'est pas affiché sur un cercle complet, mais au maximum sur un demi-cercle (Figure 4.7 page 85). On diminue ainsi les problèmes d'occlusions visuelles, qui feraient qu'un élément d'avant-plan cacherait un élément d'arrière plan. Le demi-cercle est visualisé dans sa concavité –i.e. l'observateur se trouve proche du centre du cercle. Ce point de vue est choisi pour augmenter la cohérence entre la visualisation et le contrôle de la métaphore, et favorise également une bonne vue des éléments aux extrémités. Bien sur, si l'on décidait de changer la technique de contrôle (par exemple en optant pour une rotation autour d'un axe différent), il faudrait également adapter la visualisation.

Répartition des éléments

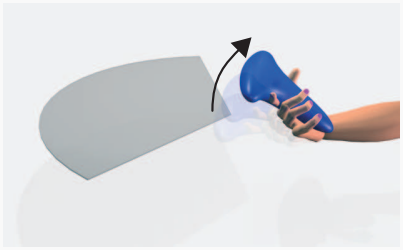
Il y a deux options pour répartir les éléments sur l'espace d'affichage, dont nous verrons les avantages et inconvénients respectifs dans la section 4.4.5.

Element de taille fixe On peut décider d'attribuer une taille fixe à chaque élément, et ajuster le périmètre de l'anneau en fonction du nombre d'éléments à placer. Notons que plus le nombre d'éléments augmente, plus l'utilisateur sera amené à faire des rotations amples pour sélectionner les éléments aux bords.

Element de taille variable On peut décider d'attribuer une taille fixe à l'anneau contenant les éléments, et subdiviser cette taille de manière à y placer tous les éléments, en la divisant en N tranches. Notons que dans ce cas, si le

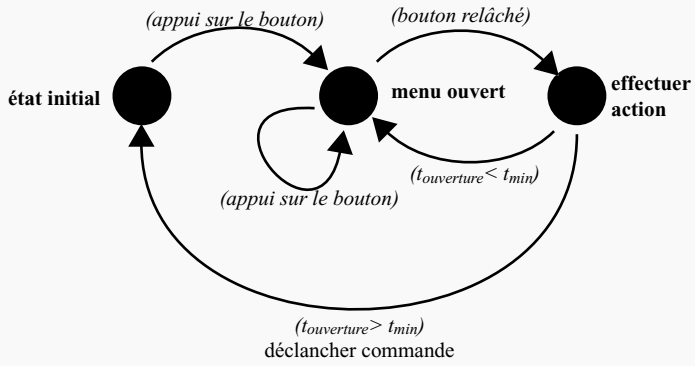


Manipulation du menu dans le plan horizontal.



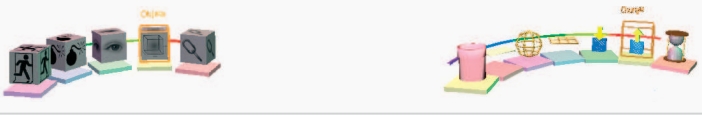
Fermeture du menu par geste dans le plan vertical.

FIG. 4.5 – Manipulation du menu



La grandeur t_{min} fait référence au seuil décrit à la section 4.7.6 page 106.

FIG. 4.6 – Principe de manipulation du Spin Menu.



A gauche, les éléments sont représentés par des boites texturées. A droite, les éléments sont représentés par des objets 3D. Les deux sont bien entendu combinables entre elles. Notons que la description de l'élément actif est toujours affichée au-dessus.

FIG. 4.7 – Représentations graphiques du Spin Menu.

nombre d'éléments devient important, la précision peut devenir problématique puisque chaque élément deviendra de plus en plus petit.

Validation de la sélection

La validation de la sélection se fait par appui et/ou relâchement d'un bouton, ce qui peut provoquer un petit mouvement parasite et causer des erreurs de sélection. Nous proposons une technique de filtrage adaptée pour diminuer sensiblement la fréquence de ces erreurs, la technique par *double intervalle*. De plus amples informations ainsi que des tests utilisateurs sont décrits dans la section 4.4.6

Hiérarchie

Enfin, nous introduisons la notion de hiérarchie dans le Spin Menu. Elle permet de créer des menus dont certains éléments ne sont pas des commandes terminales, mais ouvrent à leur tour un nouveau menu, autorisant ainsi la création d'arborescence de menus. Grâce à cette technique, classique dans toutes les métaphores de menus, nous levons –au moins en partie– la limitation du nombre d'éléments d'un menu. Évidemment, un seul menu sera toujours limité, mais on pourra éventuellement recourir à des sous-menus pour grouper logiquement certaines commandes. Nous reviendrons sur cet aspect au chapitre suivant.

4.4 Mise au point de la métaphore

Comme nous venons de l'évoquer, le Spin Menu que nous proposons ici est différent sous plusieurs aspects de la conception initiale de Liang. Nous allons maintenant voir comment chacun de ses aspects a été mis au point et validé auprès des utilisateurs.

4.4.1 Méthodologie de test

Lors de l'évaluation d'une métaphore, quelle que soit sa vocation, on peut procéder de diverses manières. Les deux plus utilisées sont les évaluations dites *sommatives* et les évaluations dites *cognitives* [BJMP01].

Une *évaluation sommative* consiste à faire passer le même test à un échantillon d'utilisateurs, de manière à mettre en évidence une caractéristique particulière d'une métaphore. De telles évaluations permettent de cibler les points forts et les points faibles de la métaphore, en ayant au préalable une idée suffisante des critères significatifs de celle-ci. Classiquement, les utilisateurs reçoivent des instructions précises qu'ils doivent suivre. L'application

sur laquelle se déroulent les tests est en général elle-même volontairement simplifiée, de manière à minimiser la dépendance des résultats à l'entraînement des utilisateurs sur l'application considérée. Ces tests ont souvent l'avantage de ne nécessiter aucun pré-requis particulier, et sont donc accessibles à (presque) tout le monde. On pourra néanmoins reprocher à ce type d'évaluations de s'éloigner parfois beaucoup d'un cadre *réaliste* d'application. En effet, le comportement ou les performances des utilisateurs sont également dépendants de la complexité de la tâche qu'ils ont à résoudre. Si celle-ci est simple, ils pourront se concentrer totalement sur la métaphore évaluée, et donc la faire apparaître comme meilleure qu'elle ne le serait en situation réelle. Pour cela, des tests sommatifs sont souvent complétés par une ou plusieurs évaluations cognitives.

Une *évaluation cognitive* consiste au contraire à laisser des experts manipuler la métaphore ou l'application, de manière à essayer toutes les tâches possibles et imaginables. Contrairement à une évaluation sommative, les utilisateurs n'ont pas de tâche particulière à exécuter, mais doivent explorer l'application ou la métaphore par rapport aux fonctionnalités qu'elle propose. Chacune d'elle est alors analysée, et l'on détermine si la ou les manipulations requises pour mener à bien la tâche forment une suite logique. Ce sont des évaluations qui font intervenir moins de monde, car elles requièrent un entraînement conséquent sur la métaphore ou l'application que l'on cherche à évaluer. De telles évaluations sont généralement suivies d'une discussion au cours de laquelle on peut débattre des points forts et des faiblesses de l'application et/ou de la métaphore.

4.4.2 Tests du Spin Menu

Sur la métaphore du Spin Menu, de nombreux tests ont été conduits auprès des utilisateurs sur divers aspects. Beaucoup d'entre eux partagent un protocole commun, que nous allons décrire ici. Les éventuelles particularités seront introduites au fur et à mesure. Si nous nous référons aux deux types de tests (cognitifs et sommatifs) décrits dans la section précédente, les tests effectués sur le Spin Menu appartiennent totalement à la catégorie sommative.

Les utilisateurs avaient pour tâche de sélectionner un cube blanc, dans un menu composé de $N-1$ cubes rouges et d'un cube blanc (Figure 4.8). A chaque ouverture de menu, le cube blanc changeait de place, chaque emplacement étant utilisé au moins une fois. La séquence était la même pour tous les utilisateurs. La taille visuelle de la métaphore permettait de voir l'intégralité du menu d'un seul coup d'œil, et donc de localiser très rapidement le cube blanc. L'utilisateur devait alors tourner le poignet pour amener le cube blanc dans la zone de sélection, aussi vite que possible ou en essayant de commettre le moins d'erreurs possibles, selon les consignes données au

préalable. Chaque test était composé de 50 tâches de sélection. Tant que le bon cube n'était pas sélectionné, la séquence n'avançait pas. Un utilisateur pouvait donc être amené à ouvrir plus de 50 fois le menu s'il commettait des erreurs de sélection.

Durant les tests, seule était comptée la durée pendant laquelle le menu était maintenu ouvert, ce qui permettait à chacun d'espacer les manipulations comme il le souhaitait, notamment pour se remettre dans une position confortable. A chaque validation d'une sélection, la durée d'ouverture du menu était enregistrée, ainsi que le nombre d'erreurs éventuellement commises jusqu'à avoir sélectionné l'élément demandé.

Les tests ont été effectués auprès d'un nombre variable d'utilisateurs, dont un bon nombre ont passé plusieurs, voire tous les tests. Les utilisateurs ont été pris sans critères particuliers. Nous avons essayé dans la mesure du possible de faire passer nos tests à quelques utilisateurs non expérimentés afin de ne pas biaiser les résultats. Lors de chaque test, nous rappellerons quels ont été les utilisateurs qui ont passé le tests ainsi que quelques statistiques les concernant. Le Tableau 4.1 donne un aperçu global des utilisateurs ayant passé les tests.

4.4.3 Choix des utilisateurs

Lorsque l'on cherche à valider une quelconque idée auprès d'utilisateurs, se pose la question du choix des utilisateurs. S'il s'agit d'un concept destiné à une catégorie d'utilisateurs particuliers, par exemple des chimistes, le panel d'utilisateurs représentatifs est d'emblée restreint. Si ce public est satisfait, on peut considérer que l'idée elle-même est satisfaisante, étant donné qu'elle convient aux personnes auxquelles elle est destinée. Pour une idée d'ordre général au contraire, destinée à un public apriori large, la question est d'autant plus importante : un mauvais choix des utilisateurs peut mener à de mauvaises conclusions.

Le Spin Menu entre dans la catégorie généraliste. Cependant, dans le contexte actuel, le public auquel s'adresse une métaphore de réalité virtuelle n'est pas aussi large que cela, en raison de la faible disponibilité de ce type d'environnement. Même si l'informatique est présente dans un nombre croissant de foyers un environnement de type *workbench* reste l'exclusivité de quelques utilisateurs privilégiés, de surcroît souvent entraînés. De ce fait, les utilisateurs que nous avons retenus sont majoritairement issus du milieu scientifique, sans pour autant être des experts de la réalité virtuelle. Quelques utilisateurs totalement novices ont également passé les tests, de même que quelques utilisateurs experts. Par experts, nous entendons ici des utilisateurs habitués à manier le *wand* pour des tâches de sélection/manipulation. Il s'agit notemment de collègues ayant fait du développement pour la réalité virtuelle. Leur expertise ne se situe néanmoins pas sur la métaphore du Spin Menu,

dont il ignoraient les spécificités et les avancées. Par opposition avec les utilisateurs experts, les novices étaient des utilisateurs n'ayant jamais manipulé d'environnement virtuel. La répartition exacte des utilisateurs pour chaque test est indiquée dans le Tableau 4.1.

4.4.4 Analyse de résultats

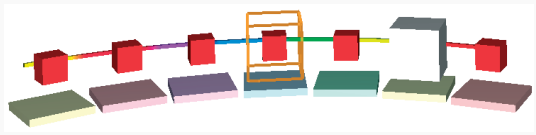
Les résultats des expériences sont souvent appuyées par une analyse de variance. Celle-ci permet de valider ou d'infirmer la comparaison entre les moyennes de différents échantillons, en permettant de quantifier la probabilité de l'hypothèse nulle dans les valeurs observées. L'hypothèse nulle correspond au fait que les différences observées dans les différentes populations ne sont dues à aucun facteur en particulier, qu'elles sont dues au hasard en d'autres termes. Le but est en général de rejeter l'hypothèse nulle, ce qui donne alors un sens aux comparaisons des moyennes. L'analyse de variance quantifie la probabilité p de l'hypothèse nulle, on considère en général qu'une hypothèse est vraie (où plutôt que l'hypothèse nulle est écartée) si $p < 0,05$, ou dans certains cas critiques (en médecine par exemple) quand $p < 0,005$.

4.4.5 De la main à la métaphore

Le Spin Menu se manipule par un mouvement de rotation (rotation physique), duquel dépend l'affichage (rotation logique). On peut intercaler une fonction de transfert entre la rotation physique et la rotation logique, de sorte à soit accélérer le feedback visuel, et donc rendre la métaphore plus sensible, soit au contraire la ralentir. La sensibilité perçue de la métaphore est directement reliée au retour visuel, puisqu'un élément devient actif lorsqu'il est en position centrale sur l'anneau (encadré dans la boîte orange).

Une fonction de transfert linéaire permettrait de rendre la métaphore uniformément plus ou moins sensible. Il est également possible, et bien plus intéressant, d'opter pour une fonction de transfert non linéaire. En effet, vu le type de mouvements que nous proposons, le poignet peut facilement arriver à sa position limite, pour laquelle la précision et le confort diminuent sensiblement, alors que la précision est supérieure lorsque le poignet est proche de sa position *neutre*, dans le prolongement du bras. On peut tirer profit de ceci pour diminuer la taille angulaire des éléments proches du centre, et augmenter celle des éléments les plus extérieurs. L'affichage peut ou non suivre les même changements. Ceci permet d'augmenter légèrement la précision de la manipulation, ou tout au moins la sensation qu'en a l'utilisateur. La figure 4.9 illustre quelques fonctions de transfert typiques, dont une non linéaire qui a pour effet d'augmenter la taille des éléments aux bords de l'anneau.

L'utilisation d'une fonction de transfert est également nécessaire pour tenir compte du fait que certaines personnes ont une vision inversée du contrôle de



L'utilisateur doit sélectionner, le plus vite possible, l'élément blanc.

FIG. 4.8 – Évaluation de la métaphore de Spin Menu.

	Participants	Age			Sexe		Expertise		
		Min	Moy	Max	♂	♀	Novice	Intermédiaire	Expert
Nombre d'éléments et répartition	9	22	27	39	6	3	3	3	3
Fonctions de filtrage	11	23	27	39	6	5	5	3	3
Main non dominante	10	23	27	39	6	4	3	3	4
Comparaison Spin, C^3, Fenêtres 3D	19	12	32	50	12	7	13	3	3
Représentation de la hiérarchie	8	23	28	39	6	2	1	3	4
Vitesse menu déroulant 2D	8	12	26	43	5	3	5	1	2

TAB. 4.1 – Répartition des utilisateurs selon les tests.

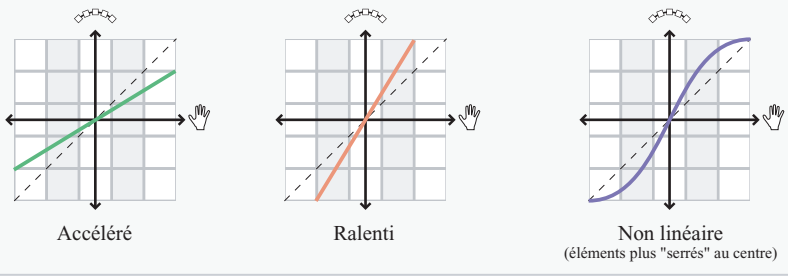


FIG. 4.9 – Différentes fonctions de transfert et leur effet perçu

l'anneau par rapport aux autres. Lorsque la main effectue une rotation vers la droite, l'anneau effectue lui aussi une rotation vers la droite, comme si la main *tenait l'anneau*. Mais puisque l'élément sélectionné reste fixe, c'est l'élément à gauche qui devient l'élément actif –alors qu'on a tourné la main vers la droite. Selon la vision que l'on a de la situation, on peut soit imaginer que la main manipule l'anneau –vision *à l'endroit* correspondant à ce qui se passe effectivement, soit imaginer que la main contrôle la sélection, ce qui est juste l'opposé. La fonction de transfert peut parfaitement être modifiée de façon à prendre en compte chacune des deux visions (soit $\alpha_l = f(\alpha_\phi)$, soit $\alpha_l = f(-\alpha_\phi)$). Dans la pratique, il s'avère qu'environ la moitié des personnes testées ont une manipulation inversée. Durant les tests que nous avons menés sur les divers aspects de la métaphore, chaque utilisateur décidait de lui-même laquelle des versions lui convenait le mieux : sa prise en main et les premières sélections étaient en général suffisantes pour déterminer quel mode était adapté.

Il reste encore un point à déterminer : comment répartir les éléments dans l'espace de manipulation ? Nous avons déja évoqué la notion d'éléments de taille fixe ou variable dans la section 84. En prenant en compte le fait que, d'après le mouvement de rotation que nous employons pour contrôler la métaphore, le poignet dispose d'une amplitude maximal d'environ 70° ([Huc93], voire figure 4.10), nous avons soit la possibilité de diviser ces 70° disonibles en N tranches (pour des éléments de taille variable), soit utiliser toujours la même taille de tranche, au risque de dépasser les 70° au dessus d'un certain nombre d'éléments (pour les éléments de taille fixe). Chacune de ces options peut être prise en charge par une fonction de transfert adaptée. On peut dores et déjà remarquer qu'en optant pour des éléments de taille variable, l'utilisateur n'effectuera pas le même mouvement pour sélectionner l'élément juste à droite de l'élément central selon le nombre total d'éléments dans le menu, l'obligeant à s'adapter à chaque menu qu'il manipule.

Détermination expérimentale

Afin de déterminer la meilleure manière d'utiliser l'espace de manipulation (éléments de taille fixe ou variable selon le nombre total d'éléments dans le menu), nous avons effectué une série de tests, basés sur la méthodologie énoncée en 4.4.1. Nous avons fait manipuler les utilisateurs sur des anneaux comportant 5, 7, 9, 11 et 15 éléments, une fois avec des éléments de taille fixe (chaque élément faisait 8 degrés, valeur que nous avons déterminée expérimentalement) et une autre fois avec des éléments de taille variable. Le même test nous a également permis de quantifier la taille maximale qu'un menu ne devrait pas dépasser, grâce aux menus de différente taille.

Les utilisateurs ont passé un ensemble de 10 tests. Aucune fonction de transfert n'était appliquée –excepté l'inversion pour certaines personnes, la manipulation et

le retour visuel étaient donc parfaitement synchronisés. L'absence de fonction de transfert était particulièrement remarquée lorsque le nombre d'éléments était faible sur un anneau de taille fixe, ou au contraire lorsque le nombre d'éléments était important. Les résultats de ces tests sont synthétisés dans les graphiques présentés en Figure 4.11 page 93.

Analyse des résultats

On peut commencer par constater que les temps de sélections sont globalement dépendants du nombre d'éléments sur l'anneau. Ce phénomène est logique, puisque plus il y a d'éléments, plus les rotations à effectuer sont amples, et donc longues. Une analyse de variance sur les temps nous confirme cette hypothèse avec une probabilité d'erreur de $p << 0,05$ pour chacune des deux familles de tests.

Avec des éléments de 8° chacun, la limite théorique serait de $70 \div 8 \approx 9$ éléments. Cependant, malgré la limité théorique annoncée, les temps ne s'effondrent pas dramatiquement entre 9 et 11 éléments. En fait, les temps moyens suivent plus ou moins une progression linéaire, alors que l'on se serait attendu à voir apparaître une nette dégradation au dessus de 9 éléments. Ceci trouve son explication dans la façon dont les utilisateurs manipulent la métaphore : le poignet est secondé par tout l'avant bras pour effectuer la rotation demandée, surtout lorsque celle-ci devient grande. La limite de 70° de rotation du poignet s'en trouve donc considérablement augmentée. Bien qu'aucune instruction n'ait été donnée en ce sens, les utilisateurs ont automatiquement adopté ce comportement.

Une différence notable apparaît entre les deux façons de répartir les éléments sur l'anneau. Il semble préférable d'adapter la taille de l'anneau au nombre d'éléments. En fixant le périmètre de l'anneau[1], les élément sont soit trop espacés (ce qui donne l'impression que la manipulation est trop «lente»), soit trop serrés (ce qui donne l'impression que la manipulation est trop «sensible»). De plus, la taille dans l'espace manipulation des éléments dépend du nombre d'éléments que le menu contient, et peut donc changer d'un menu à l'autre, obligeant l'utilisateur à s'adapter à chaque nouveau menu. Le fait de laisser le périmètre s'adapter à des éléments de taille fixe ne pose pas ces problèmes. Comme nous bénéficions en plus de la rotation du poignet, l'espace de manipulation est relativement vaste (proche de 180°), ce qui permettrait de placer de jusqu'à 23 éléments. Toutefois, les utilisateurs ont déclaré ressentir comme difficile les sélections dans des menus de 11 éléments et plus, à cause de la taille visuelle des menus, devenant difficiles à cerner d'un coup d'œil.

Nous avons par conséquent retenu de fixer la taille des éléments à 8° chacun, et d'ajuster le périmètre de l'anneau en conséquence. Le nombre maximal d'éléments contenus sur un menu, même s'il peut être plus élevé, reste limité à 9 éléments.

[1] Remarquons qu'il serait possible d'introduire une fonction de transfert qui rendrait constante la manipulation malgré l'écart visuel des éléments, mais ceci introduirait une abstraction supplémentaire.

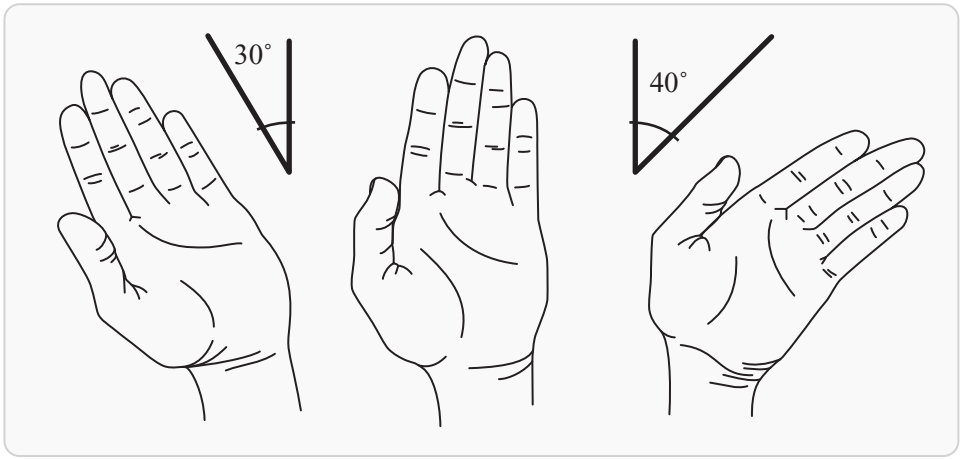


FIG. 4.10 – Paramètres orthopédiques du poignet.

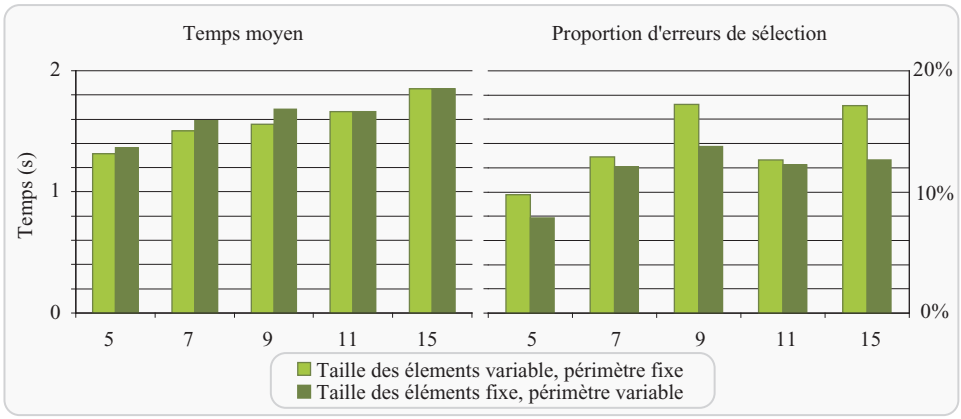


FIG. 4.11 – Temps moyen de sélection en fonction du nombre d'éléments.

4.4.6 Filtrage des mouvements

Le périphérique de pointage que nous utilisons, le *wand*, se tient en main un peu comme une épée d'escrime. La validation d'un élément se fait par l'appui sur un bouton situé sur le dessus de ce même périphérique. Malheureusement, l'appui sur le bouton peut induire un léger mouvement sur le poignet, et ainsi faire *glisser* la sélection sur l'élément voisin de celui que l'on désirait sélectionner. La Figure 4.12 page 101 illustre ce phénomène, qui est d'autant plus prononcé que l'on cherche à gagner en vitesse (dans ce cas, l'utilisateur aura parfois tendence à entammer le mouvement de retour alors que le bouton ne sera pas encore relâché). Une solution pour éviter ce problème est de dissocier la main qui effectue la sélection de la main qui la valide[SHPH04], mais ceci impose l'utilisation de deux mains pour la tâche de sélection dans le menu, que l'on souhaiterait le moins *intrusive* possible dans la manipulation.

Ce mouvement, même infime, existe. Après observation de la manipulation de nos utilisateurs, nous avons constaté qu'en règle générale, pour atteindre un élément donné, les utilisateurs minimisent naturellement l'amplitude de la rotation : dès que l'élément souhaité apparaît comme sélectionné (c'est à dire qu'il est placé dans la boite orange de sélection), ils cessent le mouvement et valident la sélection (en relâchant le bouton). Ce comportement, quoique logique, a une conséquence gênante : le poignet désigne quasiment toujours la frontière entre deux éléments. En appuyant sur le bouton, l'angle physique peut légèrement changer et «glisser» sur l'élément voisin, qui sera alors validé en lieu et place de celui souhaité si aucun traitement adapté n'est appliqué entre l'angle physique et l'angle logique.

Nous avons donc cherché à diminuer un maximum cet effet indésirable par une technique de filtrage adaptée. L'idée intuitive serait de détecter les *glissements* du poignet qui se produisent juste avant la validation, et de les supprimer. Pour cela, nous avons élaboré plusieurs techniques de filtrage, que nous avons soumises aux utilisateurs. Ces fonctions de filtrage viennent se composer avec la fonction de transfert évoquée dans le paragraphe précédent. Les différentes techniques sont les suivantes, et sont illustrées en figure 4.13[2]

Sans filtrage Les rotations issues de la fonctions de transfert sont directement transmises à l'anneau. Cette technique est incluse à des fins de comparaisons pour évaluer les gains ou les pertes de chaque autre méthode.

Filtre passe-bas L'idée de ce filtrage est de simplement effectuer une moyenne de la rotation physique de l'utilisateur sur les x dernières millisecondes, x étant un paramètre de la technique. On espère ainsi que l'effet de bord soit suffisamment gommé pour ne plus être dérangeant. Cette technique s'apparente à un filtre passe-bas, qui gommerait les "hautes fréquences" que sont les mouvements parasites. En pratique, pour que cette technique soit efficace, il faut effectuer une moyenne sur une durée trop longue, ce qui rend la métaphore

[2] Sur le graphique apparaît une technique supplémentaire, le *spin bar*, dont nous reparlerons à la section 4.6 page 99.

trop peu réactive pour être utilisable.

Magnétisme Considérons que le centre de chaque élément est un aimant, et que l'utilisateur tienne en main une barrette de fer. Celle-ci sera alors attirée par le centre de chaque élément, et l'éloignera donc d'autant du bord. Comme un aimant réel, il faut pour s'en détacher tirer suffisamment fort, ou dans notre cas s'éloigner assez du centre de l'élément pour annuler l'effet d'attraction. Cette technique cherche à faire comme si l'utilisateur se plaçait toujours au centre de la zone d'activation de chaque élément.

Filtrage à deux intervalles Imaginons que la zone de sélection de chaque élément soit divisée en deux zones différentes : une zone active et une zone inerte entre chaque élément. La zone inerte a pour propriété de ne pas permettre un changement de l'élément actif, tout en permettant de valider la sélection. La zone active permet les deux actions. De par la position des différentes zones les unes par rapport aux autres, les zones inertes jouent un rôle d'isolant, qui permettent d'augmenter virtuellement la taille de l'élément actif pendant la manipulation. En quelques sortes, on oblige l'utilisateur à s'éloigner suffisamment de la frontière entre deux éléments pour que l'éventuel mouvement parasite ne soit plus un problème (par opposition avec la méthode de *magnétisme*, qui elle ne fait que simuler cet éloignement). La technique ajoute un paramètre de plus à la métaphore : la proportion *zone active*/*zone inerte*. Nous avons utilisé dans nos test une proportion de 50%. En fait, la proportion exacte n'est pas d'une importance capitale, car même une zone active extremement petite permettrait de sélectionner l'élément, même si l'angle logique ne fait que *traverser* la zone.

Filtrage à deux intervalles amélioré Ce filtrage est une variante de la technique précédente, qui tente de diminuer à nouveau l'amplitude de la rotation demandée, en déplaçant les zones actives de chaque élément vers le centre de l'anneau. Le principe des zones actives et inertes reste identique, seul leurs positions changent.

Evaluation des filtrages

Ces fonctions de filtrage offrent différents degrés de correction, mesurés par deux grandeurs : la performance moyenne de sélection en termes de temps, ainsi que le nombre d'erreurs de sélections qui passent au travers de la technique de filtrage. Ces critères sont évalués sous deux conditions différentes : soit en mettant l'accent sur la performance, au détriment de la précision, soit au contraire en mettant l'accent sur la précision au détriment de la performance. Ces deux extrêmes capturent les comportements possibles des utilisateurs. Les performances réelles de la métaphore, en utilisation dans une application complexe, oscillent entre ces deux extrêmes : pour les commandes utilisées fréquemment, l'utilisateur s'autorise à manipuler rapidement, alors que pour les commandes peu usitées la vitesse diminue automatiquement et la concentration augmente.

Ces différentes fonctions de filtrage ont été évaluées selon la méthodologie décrite au paragraphe 4.4.1. Chaque utilisateur a effectué chaque test deux fois, pour mettre l'accent sur la précision ou la vitesse. Les résultats sont synthétisés dans le tableau 4.14 page 103.

On peut voir que la méthode de *filtrage à deux intervalles* (amélioré ou non) apporte la meilleure correction, principalement lorsque l'on cherche justement à éviter les erreurs. Une analyse de variance permet de valider le fait que la technique de filtrage a une incidence sur les temps de sélection avec une probabilité d'hypothèse nulle de $p = 0,009$ sous la condition d'aller le plus vite possible et $p = 0,01$ sous la condition d'être le plus précis possible. Une analyse de variance entre l'absence de filtrage et le filtrage par double intervalle donne des probabilité d'hypothèse nulle de $p = 0,02$ et $p = 0.007$ selon les deux modalités de test. Les probabilité d'hypothèse entre l'absence de filtrage et la technique à double intervalle décentrés est de $p = 0.004$ et $p = 0.02$ selon les deux modalités de test. Durant la manipulation, les utilisateurs ont souvent constaté oralement que cette technique *répondait bien*, et causait moins de fausses sélections auxquelles ils ne s'attendaient pas. Pour les autres techniques, même si elles diminuent légèrement la proportion d'erreurs, les utilisateurs restaient souvent perplexes devant le nombre important d'erreurs qu'ils commettaient malgré tout : il n'était pas rare d'entendre des remarques comme «je n'avais pourtant pas l'impression de m'être trompé, là !». Il serait par contre faux de croire que la technique par double intervalle décentrés est réellement meilleure que la technique à double intervalle car la probabilité d'hypothèse nulle est de $p = 0.44$ et $p = 0.82$ selon les deux modalités de tests. On peut donc en utiliser l'une ou l'autre, les différences étant dues en bonne partie au hasard.

On peut noter également, que tout en améliorant sensiblement le nombre d'erreurs, les fonctions de filtrage améliorent également les performances. Il s'agit là d'une conséquence logique : puisque la métaphore semble plus fiable, on tend naturellement à accélérer la cadence. Même si ce n'était pas le but des fonctions de filtrage, il s'agit là d'un effet de bord appréciable.

4.4.7 Main non dominante

La manipulation requise par la métaphore est relativement simple. Ceci nous a suggéré d'évaluer la métaphore en optant pour une manipulation avec la main non dominante.

Nous avons effectué un test simple, sur le même principe que pour l'évaluation des fonctions de filtrage, excepté que les utilisateurs devaient manipuler uniquement avec leur main non dominante. Seul le filtrage à double intervalle a été utilisé (la main non dominante étant a priori moins précise que la main dominante, il est préférable d'opter pour la technique de filtrage la plus efficace). Les résultats de ce test sont présentés schématiquement dans le tableau 4.2. On peut noter que ni les performances ni le taux d'erreurs n'augmentent de façon significative. De façon surprenante, c'est le contraire qui semble se produire : les résultats sont, en moyenne, légèrement meilleurs. Une analyse de variance entre les deux modes de

manipulation confirme qu'il s'agit bien d'un fait avec une probabilité d'hypothèse nulle de $p = 0.02$ en mode précision, et $p = 0.005$ en mode rapidité. Nous n'avons pour l'instant aucune explication à ce phénomène.

4.5 Comparaison expérimentale

A ce stade, nous disposons d'une métaphore parfaitement opérationnelle, quoique limitée en nombre d'éléments. Nous avons jugé opportun d'évaluer cette métaphore par rapport à d'autres métaphores déjà existantes qui adressent la même problématique. La littérature ne compte que peu de métaphores de contrôle d'application, et d'autant moins de métaphores dédiées aux menus, comme nous l'avons vu dans l'état de l'art. Nous avons considéré trois métaphores avec lesquelles comparer le Spin Menu, mais n'en avons finalement retenues que deux. Les raisons de ces choix sont détaillés ci-dessous.

Fenêtres 3D Les fenêtres 3D représentent la transposition immédiate des interfaces auxquelles quasiment tout le monde est habitué, dans le monde tridimensionnel de la réalité virtuelle. Dans ce contexte, nous appelons fenêtre 3D un *espace conteneur*, généralement de forme rectangulaire, que l'utilisateur peut placer librement dans l'espace en position comme en orientation en employant la métaphore du rayon laser virtuel. Dans cet espace conteneur se trouvent des boutons rectangulaires, que l'on active en les désignant à l'aide du laser virtuel et en appuyant sur un bouton lorsqu'ils sont désignés. D'autres composants, comme des potentiomètres[3], peuvent être utilisés. La manipulation fait donc intervenir une certaine forme de précision, mais aussi de capacité à maintenir une position et une orientation le temps d'un appui sur un bouton. Cette métaphore a été retenue car elle apparaît comme une métaphore très générique, adaptable à volonté et surtout proche de ce que nous connaissons.

Command & Control Cube Cette métaphore est par définition une métaphore de menu. Il est donc logique de comparer le Spin Menu à cette métaphore. De plus, le C^3 a été élaboré sur un environnement de type workbench, donc tout à fait comparable à l'environnement utilisé pour mettre au point le Spin Menu.

Menu TULIP Le menu TULIP est lui aussi par conception une métaphore de menu. Malheureusement, nous n'avons pu l'implémenter sur notre environnement pour des raisons de physiques inhérentes au dispositif. En effet, le menu TULIP requiert la capacité de pouvoir afficher des données a proximité des mains de l'utilisateur, et souvent même entre les yeux et les mains de celui-ci. Si ceci est possible lorsque les mains de l'utilisateur sont représentées par des avatars virtuels, un workbench ne permet pas ce type d'af-

[3] Élément graphique permettant la sélection d'une valeur dans un intervalle. Certains de ces composants offrent également la possibilité de définir des bornes minimum et maximum.

fichage, puisque quoi que l'on fasse, l'écran sur lequel les images sont projetées ne peuvent être placés entre les mains et les yeux de l'utilisateur, qui occultent donc toujours l'affichage (cf. Figure 1.25 page 40).

Interface classique 2D Les interfaces WIMP sont actuellement le paradigme favori des interfaces utilisateur de nos ordinateurs habituels. De ce fait, les utilisateurs sont tous plus ou moins habitués à les manipuler, et leur apprentissage n'est plus à faire. Ce type d'interface apparaît comme une sorte de *plus petit dénominateur commun* en terme d'habitude et nous l'imaginons, de performance. Nous avons fait passer à nos utilisateurs le même type de tests sur un ordinateur équipé d'une souris comme périphérique d'entrée et de Microsoft Windows^{XP} comme interface utilisateur. Deux tests ont été faits sur cette plateforme : une représentation par boutons, et une représentation par menu. Davantage d'informations sont données dans la section suivante.

Mise en place

Afin de permettre une comparaison entre ces différentes techniques de menu, nous avons adapté le test classique déjà évoqué au paragraphe 4.4.2 aux différentes techniques retenues. Le but de notre test étant de comparer les métaphores en termes de performances maximales et de nombre d'erreurs, nous avons essayé de placer chaque métaphore dans ses propres conditions optimales. Le test classique de performance que nous avons utilisé jusqu'à présent a été adapté aux différentes métaphores. Ci-dessous sont données les différentes informations et adaptations apportées au test pour chacune des situations, représentées en Figure 4.15 page 103.

Le Spin Menu a été testé sur un anneau composé de 9 éléments (sa limite recommandée), en appliquant la technique de filtrage par double intervalle.

Le C^3 a été testé sur 26 éléments (nombre maximal d'éléments qu'il peut contenir, sur un niveau de hiérarchie), avec retour visuel uniquement. Durant le test, l'ensemble du menu était vide, sauf une case, contenant le cube blanc à sélectionner. Celui-ci était bien entendu visible quel que soit la tranche dans laquelle il se trouvait. En fait, il s'agit du même test que celui utilisé par les auteurs de la métaphore [GC01].

Les fenêtres 3D ont été testées sur $5*5$ boutons, de 10cm chacun, espacés de 3cm. Cette disposition des boutons permet, sur l'environnement utilisé, de remplir la quasi-totalité de l'écran en ayant des boutons suffisamment larges. Durant le test, tous les boutons étaient rouges, sauf un blanc et très légèrement plus gros, que l'utilisateur devait sélectionner en pointant dessus à l'aide du *wand*. Le rayon du laser virtuel était bien sur affiché comme nous le faisons toujours.

Les interfaces 2D ont été testé deux fois : en proposant un arrangement de 5x5 boutons, comme les fenêtres 3D, et en sur un menu déroulant standard composé de 7 éléments. Le test composé de 25 boutons est la réplique, en 2D, du

test des fenêtres 3D. Tous les boutons étaient vierges, sauf un marqué d'une croix. C'est celui-ci qu'il fallait cliquer, en utilisant une souris standard. La version à menu déroulant consistait en l'appui sur un bouton, qui ouvrait en dessous de lui un menu déroulant, dont l'élément à sélectionner était affiché en gras.

Résultats

Nous avons fait passer 19 utilisateurs, de profils variés, sur ce test. Les résultats indiqués par le tableau 4.16 indiquent plusieurs choses. D'abord, on peut constater –sans grande surprise– que le test 2D avec boutons offre les meilleurs performances. En revanche, il est plus surprenant de constater que le classique menu déroulant offre des temps relativement moyens (comparé aux boutons 2D), alors qu'il est considéré comme une «excellente» solution[4]. Ceci dit, il ne faut pas oublier que le but du test était d'aller le plus vite possible –quitte à commettre quelques erreurs supplémentaires. Dans la manipulation quotidienne, l'accent n'est pas autant mis sur la vitesse pure, mais plutôt sur la connaissance par coeur de l'interface par l'utilisateur.

Venons-en maintenant aux trois métaphores de réalité virtuelle. En appliquant une analyse de variance entre les trois métaphores, on peut affirmer sans nul doute qu'il existe une différence entre les métaphores ($p = 6.3e - 40$). Il est clair que le C^3 est plus lent que les fenêtres 3D ou le spin menu. Si nous nous intéressons maintenant uniquement à la comparaison entre les fenêters 3D et le spin menu, une analyse de variance valide l'affirmation selon laquelle elles sont en moyenne plus lentes que le spin menu, avec une probabilité de $p = 0.02$. Les données utilisées pour ces analyses sont fournies dans l'annexe B.3.1.

4.6 Arrangement circulaire de boutons

Wesche et Droske [WD00] proposent une variante de menu circulaire intéressante : les éléments du menu sont disposés sur un arc de cercle, mais la sélection de l'élément voulu se fait à l'aide d'un laser virtuel. Nous avons trouvé intéressant de comparer cette technique au spin menu, puisque nous utilisons nous-même le laser virtuel dans nos applications.

Nous avons implémenté cette métaphore à l'aide de la libSelect (voir chapitre 5), et fait passer les mêmes utilisateurs que pour le test sur les fonctions de filtrage. Afin d'optimiser la précision de la sélection, la zone sensible des boutons était plus grande que ce que l'utilisateur voyait réellement (5 fois plus haute). La figure 4.18 illustre la disposition des éléments ainsi que leur zone sensible.

[4] Si l'on en croit les adeptes des menus circulaires, le menu linéaire aurait du disparaître depuis longtemps.

La figure 4.3 montre les résultats comparés au spin menu[5] –dans sa version *optimale*– et au C^3. Il apparaît que la technique de Wesche et Droske donne d'excellentes performances Il parait cependant délicat de l'appliquer à la main non dominante, sauf à contraindre les mouvements du laser, auquel cas la manipulation rejoint celle du Spin menu.

4.7 Observations

La métaphore du spin menu a été principalement testée dans le cadre du prototype Dogme^{RV}. Celui-ci, proposant un nombre intéressant d'options de nature très différentes les unes des autres nous a permis de confronter la métaphore à diverses situations. Nous allons ici décrire les divers aménagements que nous avons effectués, ainsi que quelques remarques d'ordre plus général sur la métaphore elle-même.

4.7.1 Élements collants

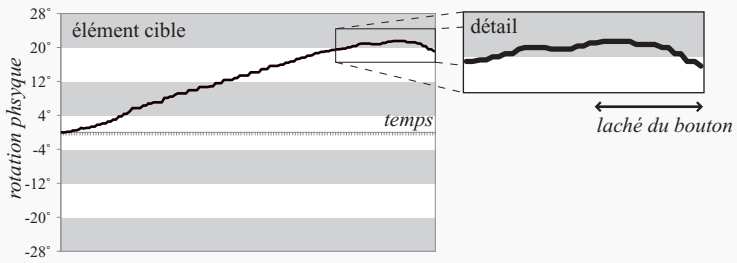
Le comportement classique d'un menu est de se refermer automatiquement dès qu'un élément est activé. Cependant, pour certains éléments, il nous a paru intéressant de laisser le menu ouvert afin de permettre un envoi rapide d'une commande proche de celle activée. Notamment, les différents modes d'affichage sont chacun des *éléments collants* : chaque mode peut être soit activé soit désactivé, indépendamment des autres. Par conséquent, si l'on souhaite passer du mode *filaire* au mode *rempli*, il faudra activer deux éléments : celui du mode rempli, qui s'ajoutera alors au mode filaire déjà actif, puis supprimer le mode filaire. Si malgré tout l'utilisateur désire n'envoyer qu'une commande, il pourra fermer le menu en utilisant le geste dédié à cet effet.

En pratique, c'est souvent à l'utilisation que l'on remarque que certains éléments ont tendance à être utilisés en séquence, et que l'on peut donc les rendre collants pour accélérer l'exécution de la séquence. Le principe n'est pas novateur en soi (par exemple, le célèbre gestionnaire de fenêtres *Window Maker* propose un mécanisme similaire permettant de rendre un sous-menu entier collant en cliquant simplement dans son titre). D'autres interfaces proposent une icône en forme de punaise pour l'on peut activer afin de laisser un élément toujours visible («l'accrocher à l'écran»).

4.7.2 Représentation des éléments

Classiquement, un menu est composé de petits labels de texte indiquant l'action qu'ils effectuent. Dans certains cas, il est plus explicite d'afficher une icône à la place d'un texte, ou en plus du texte. Par exemple, dans le menu permettant de

[5] La métaphore de Weshe et Droske est nommé *spin bar* dans les graphiques, par analogie avec l'opérateur de négation de l'algèbre booléenne parfois dénommé *barre*, puisque les éléments ne tournent pas.



Enregistrement de l'angle physique au cours du temps, dans la tâche de sélection d'un élément donné. L'utilisateur cesse tout mouvement aussitôt que l'élément apparaît comme sélectionné, et l'action de relâchement du bouton le fait glisser dans la zone d'activation de l'élément précédent.

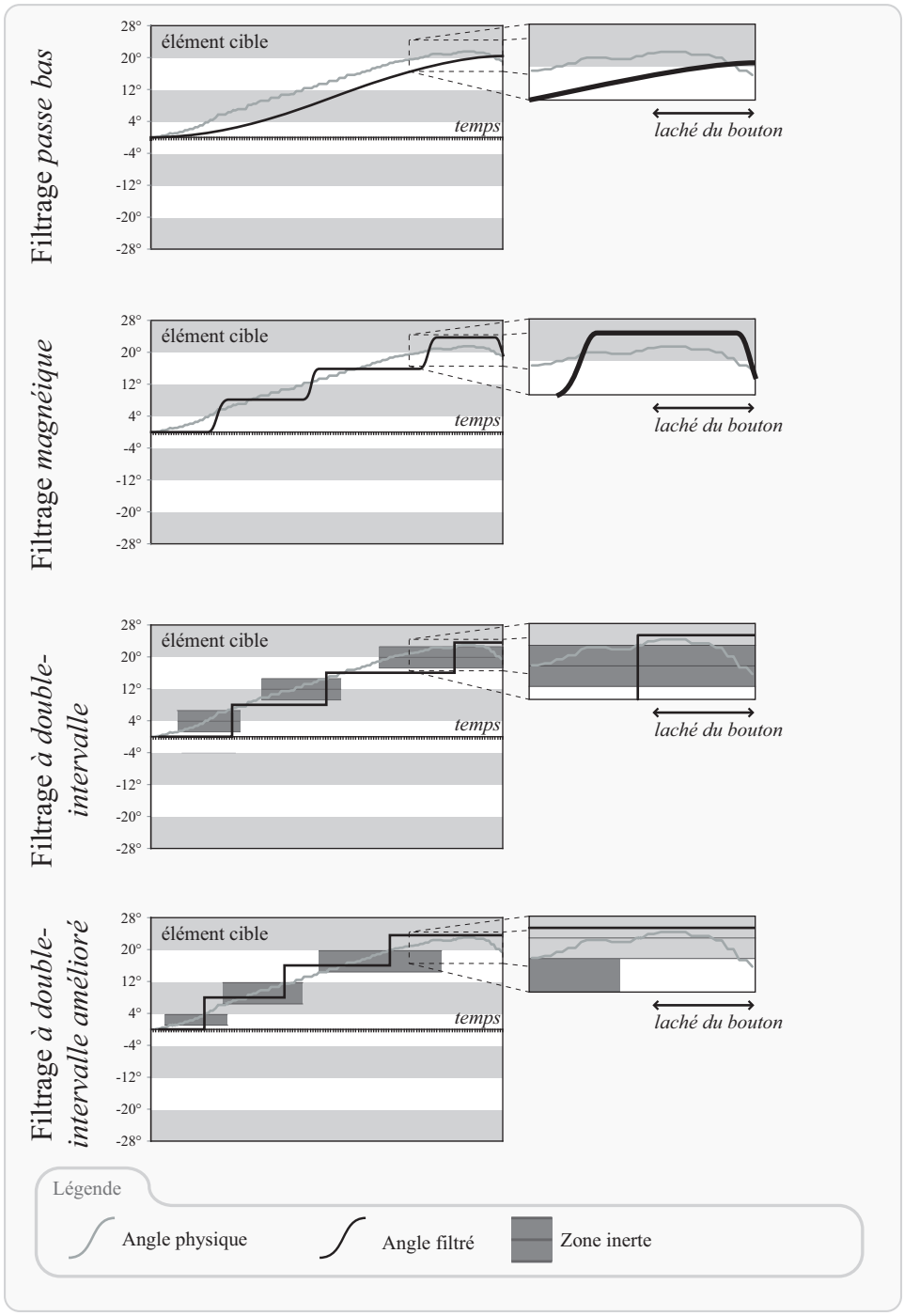
FIG. 4.12 – Mouvement de rotation parasite dû au lâché du bouton.

	Précision		Vitesse	
	Temps	Erreurs	Temps	Erreurs
Main dominante	1,51s	1,84%	1,24s	7,14%
Main non dominante	1,40s	2,65%	1,16s	5,31%

TAB. 4.2 – Comparaison main dominante/main non dominante

	Menu 2D	Boutons 2D	Fenêtres 3D	C^3	Spin Menu	Spin Bar
Temps	0.88	0.77	1.14	1.62	1.09	1.00
Erreurs	1.60%	0.31%	2.69%	7.24%	8.33%	2.81%

TAB. 4.3 – L'anneau laser par rapport aux autres métaphores.



Les différentes techniques de filtrage, appliquées à l'enregistrement de la rotation d'un utilisateur (illustration).

FIG. 4.13 – Techniques de filtrage

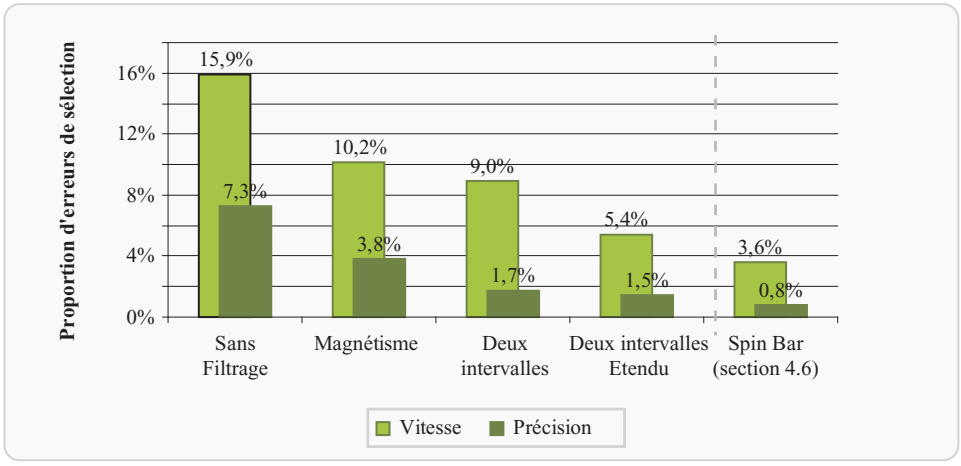
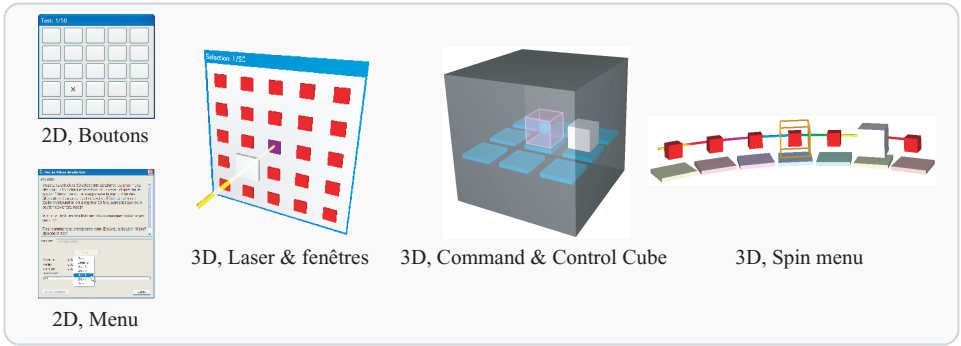


FIG. 4.14 – Résultat des différentes méthodes de filtrage.



(a) Boutons 2D (5x5 boutons). L'élément à sélectionner est marqué par un X.

(b) Menu 2D (7 éléments). L'élément à sélectionner est affiché en gras.

(c) Fenêtres 3D (5x5 boutons). L'élément à sélectionner est plus grand et blanc.

(d) C^3 (26 éléments). Seul l'élément à sélectionner est affiché.

(e) Spin Menu (9 éléments). L'élément à sélectionner est plus grand et blanc.

FIG. 4.15 – Différentes interfaces pour le même test

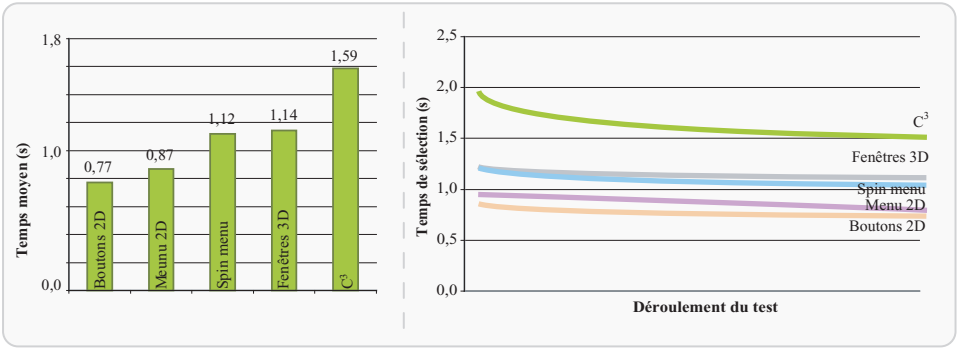
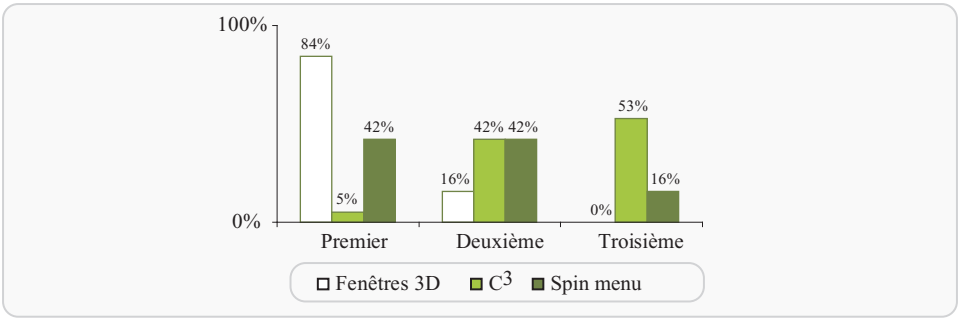


FIG. 4.16 – Comparaison des performances de sélection entre le spin menu et d'autres métaphores classiques.



Proportion, pour chaque métaphore, d'avoir été classée première, deuxième, ou troisième par les utilisateurs.

FIG. 4.17 – Le spin menu face à d'autres métaphores.

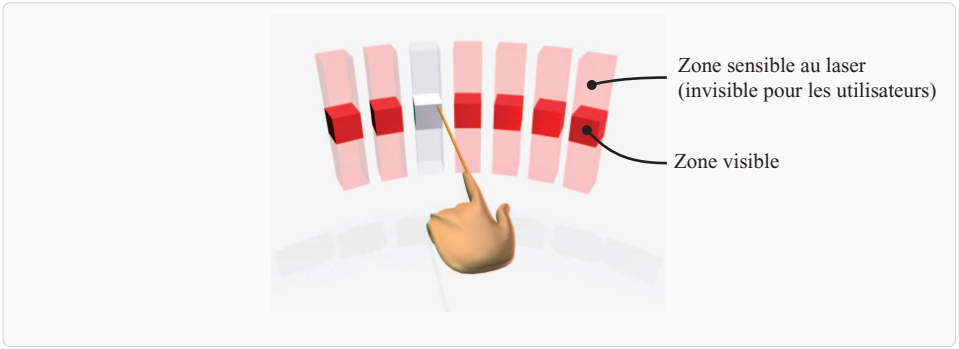


FIG. 4.18 – L'arrangement de boutons circulaire de Weshe et Droske

charger des objets, une miniature de l'objet est souvent plus parlante que son nom, même si celui-ci peut aider à différencier des objets qui se ressemblent. Aussi, nous trouvons judicieux d'utiliser des icônes augmentées d'un texte descriptif. Les éléments prennent ainsi moins de place (le texte n'est affiché au dessus d'un seul élément à la fois), et les icônes peu explicites bénéficient de la présence du texte. Dans un environnement 2D, les *info-bulles* jouent le même rôle de complément d'information textuel sur une icône ou un symbole.

4.7.3 Occlusion

Au fur et à mesure que la scène se remplit, le menu a de plus en plus de chances d'entrer en collision avec des objets déjà présents. Ceci est à éviter à tout prix, puisque le menu devient par la même occasion au moins partiellement invisible. Les utilisateurs ont tendance à éviter cette situation, et referment immédiatement le menu pour l'ouvrir ailleurs lorsque cette situation se produit. Une solution facile qui vient immédiatement à l'esprit est de garantir que le menu sera toujours affiché devant les objets de la scène (en ne faisant par exemple pas de test de profondeur lors de l'affichage du menu et en l'affichant en tout dernier). Malheureusement, si ceci fonctionne bien en 2D, la stéréovision rend délicate ce genre d'artifices : si un objet «lointain» couvre un objet plus proche, la sensation de profondeur est cruellement diminuée, car le cerveau ne se dupe pas aussi facilement. Une autre solution consiste à effacer localement l'écran à l'endroit où le menu doit être affiché. On évite ainsi la mauvaise perception stéréo, mais un «trou» apparait au beau milieu de la scène. L'effacement peut se faire graduellement sur les bords du rectangle, donnant alors l'illusion d'une zone non éclairée, un peu moins surprenante qu'un trou noir surgi de nulle part. On peut également temporairement n'afficher que le menu et supprimer tout autre objet, comme le faisaient [Dee95]. Cette dernière version reste cependant déroutante pour l'utilisateur qui a l'impression d'être téléporté à un endroit différent à chaque fois qu'il manipule le menu. On peut noter que le même type de problèmes apparaît avec tout type de métaphore contextuelle, que ce soit un menu déroulant ou un C^3.

Une autre possibilité serait d'afficher la métaphore à proximité de ses yeux, comme les informations de vol sur les casques d'aviation militaire. En quelque sorte, ce serait de la réalité augmentée virtuelle. On évite ainsi la quasi-totalité des problèmes d'occlusion –à moins que l'utilisateur ne plonge la tête à l'intérieur d'un objet. Cependant, cette technique n'est pas utilisable car elle oblige à changer constamment la focalisation des yeux sur des objets (très) proches ou lointains. Une fatigue oculaire s'installe très rapidement. De plus, tous les utilisateurs ne sont pas capables de fusionner des images aussi proches : si la perception que l'on a des images est d'être affichées à quelques dizaines de centimètres des yeux, les images stéréo sont elles séparées de plusieurs dizaines de centimètres. La technique va à l'encontre du principe généralement admis qu'il faut éviter de placer des objets virtuels à moins d'un tiers de la distance entre l'utilisateur et l'écran pour des raisons de confort et de capacité de fusion des images. Nous avons également essayé de

n'afficher la métaphore que sur un oeil, sans plus de succès.

En résumé, il n'y a pas de solution miracle à ce problème. Étant donné que c'est l'utilisateur qui décide de l'emplacement où le menu va s'ouvrir, libre à lui de trouver un emplacement correct qui lui convienne.

4.7.4 Élements rapides

La vitesse de sélection d'un élément est dépendante de sa position sur l'anneau, puisque chaque élément doit être atteint pour être sélectionné, alors que la distance angulaire à parcourir est différente pour chaque élément. On conçoit aisément que l'élément central par exemple, sur lequel s'ouvre le menu, soit l'élément pour lequel le temps moyen est le plus faible. Néanmoins, on peut constater d'après les manipulations que d'autres éléments fournissent d'excellentes performances, tant en terme de vitesse qu'en terme de nombre d'erreurs. Le graphique présenté en Figure 4.19 page 108 illustre ces résultats. On peut noter la présence de trois minimums locaux : au centre, et également aux deux extrémités de part et d'autre de l'anneau. Ceci trouve son explication dans le fait que la sélection d'un élément extrémal peut se faire rapidement en allant "plus loin" que nécessaire, car la métaphore bloque la sélection si celle-ci dépasse les limites autorisées. Les utilisateurs savent alors que lorsque l'élément à sélectionner se trouve sur l'une des extrémités, la précision ne joue plus aucun rôle s'ils font un geste suffisamment ample, ce qu'ils font alors très rapidement.

4.7.5 Inversion du contrôle

Un autre aspect surprenant, dont nous avons déjà parlé dans la section sur les fonctions de transfert, est l'inversion du contrôle de certains utilisateurs. En effet, si certains utilisateurs (environ la moitié) imaginent «tenir l'anneau en main», d'autres imaginent «devoir déplacer la sélection», ce qui résulte en des mouvements exactement contraires pour atteindre un même but. Nous n'avons aucune explication à ce phénomène, qui n'influe heureusement pas sur la possibilité d'utiliser efficacement le menu. Utiliser l'un ou l'autre sens revient simplement à inverser la fonction de transfert pour prendre en compte le changement. Dans les tests que nous avons eu l'occasion de faire, ainsi que durant les manipulations de Dogme^{RV}, les quelques premiers mouvements suffisent à déterminer si un utilisateur fait partie de l'une ou l'autre catégorie. La métaphore était alors ajustée en conséquence.

4.7.6 Clic flash

A l'origine, nous avions conçu la métaphore pour rester affichée tant que le bouton d'action restait appuyé. Le relâchement du bouton provoquait l'activation de l'élément sélectionné. Mais à l'utilisation, il s'est avéré que beaucoup d'utilisateurs voyaient le bouton comme un «appel du menu», et le relâchaient dès que

celui-ci apparaissait, activant involontairement l'élément central. Cependant, les utilisateurs plus entraînés apprécient le fait de manipuler avec le bouton enfoncé. Afin de permettre l'une ou l'autre manipulation, nous avons introduit la notion de «clic flash».

La durée entre l'appui et le relâchement du bouton est mesurée. Si cette durée est inférieure à un seuil, on considère que l'utilisateur souhaite manipuler la métaphore avec le bouton relâché, le menu reste donc visible. Au contraire si la durée dépasse le seuil, on considère que l'utilisateur a eu le temps de faire sa sélection, et qu'il faut par conséquent activer la commande choisie et fermer le menu. La détermination du seuil est issue des données expérimentales des tests de rapidité. En effet, aucun utilisateur n'a fait de sélection en moins de 0.6 secondes. Par précaution, nous avons fixé le seuil à 66% de cette valeur, soit 0.4 secondes. La manipulation s'adapte ainsi dans la pratique parfaitement aux deux situations.

Cette fonction n'est pas à confondre avec le fait de pouvoir effectuer une sélection bouton relâché ou enfoncé dans un menu déroulant 2D. En effet, dans un environnement 2D, ce n'est pas le temps qui entre en compte, mais la position du curseur par rapport au menu, pour déterminer si le lâché du bouton est à interpréter comme une sélection ou non[6].

4.8 La hiérarchie

Nous l'avons déjà remarqué, le nombre d'éléments que peut contenir un anneau est limité et relativement faible. Même si ceci peut ne pas être un problème lorsque l'on parle d'un menu isolé, la situation change si les menus représentent le noyau du contrôle d'une l'application. La solution classique à ce problème est de recourir à la hiérarchie, ce qui dans le cas du spin menu revient à créer des anneaux et des sous-anneaux. La capacité totale en nombre d'éléments est ainsi augmentée, bien que le cas où un seul menu contient trop d'éléments ne soit pas réglé. On pourrait dans ce cas diviser le long menu en plus petits morceaux, reliés entre eux par la hiérarchie, mais ce n'est pas une pratique courante dans les interfaces utilisateur.

La notion de hiérarchie dans le spin menu reprend la version simple, en étendant la manipulation pour permettre une navigation au sein de cette hiérarchie, comme l'illustre l'automate présenté en Figure 4.20 page 108. Similairement, la représentation graphique doit tenir compte de la présence d'une hiérarchie.

4.8.1 Adaptation de la version simple

La notion de hiérarchie se greffe au dessus de celle de menu. En effet, elle réutilise le concept de menu pour proposer davantage d'éléments en créant un arbre ou

[6]Les menus des interfaces 2D s'ouvrent généralement légèrement décalés du curseur. En lâchant le bouton sans effectuer de déplacement –rapidement ou non, l'utilisateur n'a donc pas cliqué sur le menu, mais légèrement à coté. Le reste alors ouvert

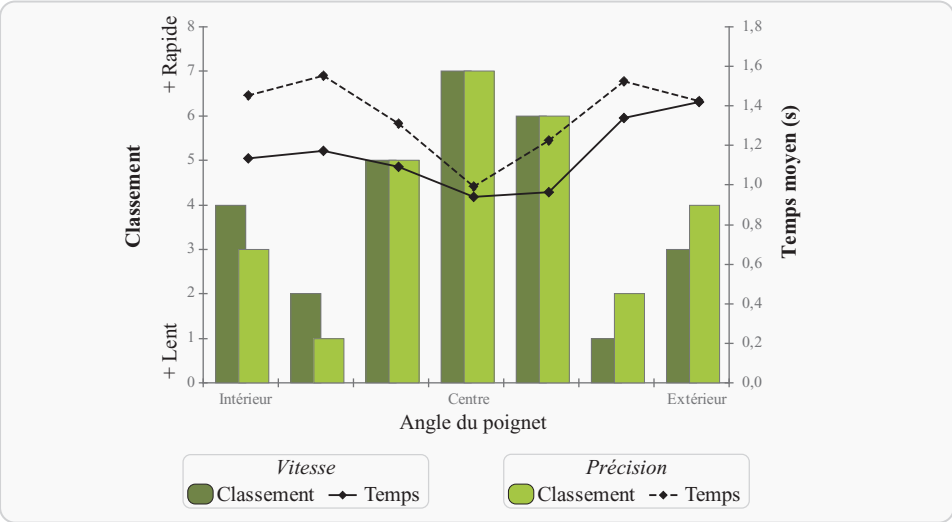


FIG. 4.19 – Classement des temps de sélection en fonction de la position de l'élément sur l'anneau.

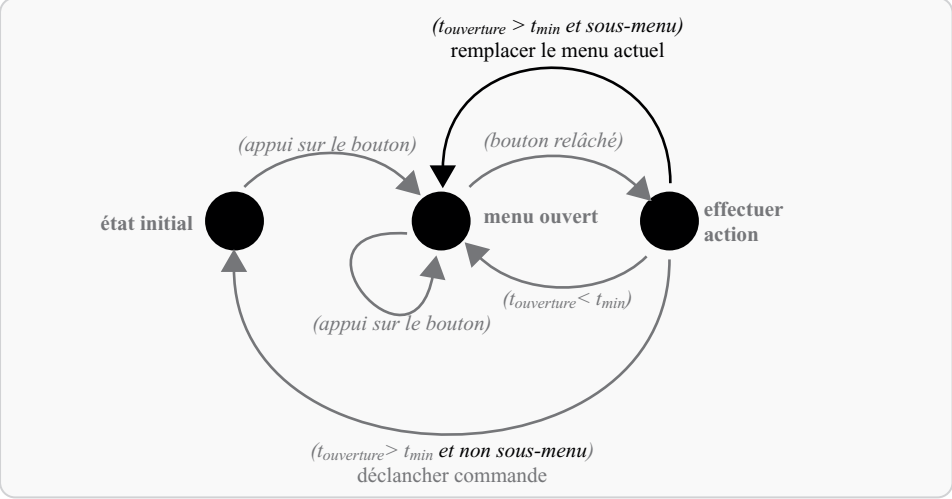


FIG. 4.20 – Principe de manipulation du spin menu hiérarchique.

un graphe (de préférence acyclique) de commandes. Ceci implique qu'il faut pouvoir *naviguer* dans cette hiérarchie de menus : certains éléments permettront de passer d'un niveau à l'autre. Visuellement, ces éléments sont identiques à des éléments *terminaux* –les commandes, sauf qu'ils sont surmontés d'une pointe orientée vers le bas, signifiant que l'on peut «descendre» dans cet élément.

De manière optimale, le chemin de navigation dans la hiérarchie devrait être le plus court possible entre le point d'entrée et l'élément à activer. Il peut cependant arriver qu'il faille faire marche arrière, soit parce qu'un mauvais menu a été ouvert, soit parce que l'utilisateur lui-même tâtonne à la recherche de l'élément qu'il désire. Il faut donc, en plus de pouvoir entrer dans un menu, pouvoir en ressortir *sans* activer d'élément. Nous pouvons envisager plusieurs solutions pour répondre à ce besoin. Par exemple, réserver un élément, éventuellement légèrement à l'écart, permettant de «remonter» d'un cran. Nous n'avons pas retenu cette idée car elle utilise un élément alors que leur nombre est déjà limité. Nous avons plutôt opté pour l'utilisation d'un geste particulier, le geste *d'annulation*, qui est le même que le geste de fermeture du menu non hiérarchique (décrit page 82). Ce geste est totalement lié à la façon dont la métaphore est contrôlée ; si ce contrôle changeait, le geste serait à adapter également. Dans la configuration proposée, ce geste est un mouvement de rotation orthogonal au plan de manipulation de l'anneau : l'utilisateur doit effectuer une rotation «vers le haut» ou «vers le bas» d'une amplitude suffisante pour fermer le menu courant et remonter au menu précédent (intuitivement, il «montre le sol» ou le plafond). Si ce geste est effectué sur le menu racine –i.e. celui qui n'a pas d'ancêtre, alors l'ensemble du menu est fermé et l'utilisateur peut reprendre son activité comme s'il n'avait jamais ouvert le menu. Il survient néanmoins un petit problème lorsque l'utilisateur désire fermer l'ensemble du menu lorsqu'il se trouve dans un sous-menu : il doit fermer chaque étage en séquence. Il peut pour cela tirer profit du fait que le geste de fermeture peut être fait vers le haut ou le bas, et «secouer» simplement la main pour fermer efficacement l'ensemble des menus ouverts.

Quand un élément terminal –commande– est activé, l'ensemble de la hiérarchie de menus est fermée, sauf pour les éléments collants. C'est le comportement par défaut de la métaphore, exactement comme pour les classiques menus 2D par exemple.

4.8.2 Représentation de la hiérarchie

La représentation de la hiérarchie est une partie importante de son utilité car plus les menus deviennent complexes, plus l'utilisateur risque de s'y perdre. Il faut donc lui donner plus d'information pour l'éviter. Dans des cas extrêmes, on peut citer certains appareils portatifs (baladeurs, téléphones mobiles, etc.) qui n'affichent qu'un seul élément sur leur écran, sans aucune trace de la hiérarchie. Les contraintes techniques excusent ces concessions, mais l'utilisateur a tout intérêt à connaître les menus pour s'en sortir. Dans un environnement virtuel, nous n'avons a priori aucune contrainte de ce type, nous pouvons donc imaginer diverses re-

présentations, plus ou moins efficaces, pour donner à l'utilisateur un maximum d'information sur sa position dans la hiérarchie et le chemin qu'il a emprunté pour y parvenir. Ceci lui permettra d'une part d'apprendre plus rapidement la hiérarchie de menus par l'utilisation, et surtout lui permettra de retrouver aisément où il se situe s'il devait lui arriver de se perdre dans les menus –ce qui arrive plus souvent que l'on ne pense.

De nombreux sites web, tels que le moteur de recherche Yahoo^{TM}, proposent une vision classée (donc hiérarchique) d'un grand nombre d'éléments, affichent un *chemin inverse* (anglais *back path*) qui représente le chemin qu'a emprunté l'utilisateur pour aboutir à la page qu'il visualise. Il peut ainsi aisément remonter dans la hiérarchie si la page qu'il obtient ne correspond pas à ses critères, ou s'il désire explorer des pages connexes. Puisque nous savons a présent que nous devons proposer à l'utilisateur une vision claire de sa position dans la hiérarchie de menus, voyons comment graphiquement la dessiner.

Une façon qui semblait simple et efficace est la suivante. Souvenons-nous que la métaphore est basée sur la rotation d'une portion de cercle autour d'un axe. Par conséquent, nous disposons d'un point fixe par rapport à l'anneau en rotation : son axe de rotation. On pourrait donc placer le long de cet axe les différents éléments qui forment le chemin de navigation de l'utilisateur dans la hiérarchie. Cet axe est intéressant, puisqu'il est connecté de manière logique à l'anneau, et reste naturellement insensible à la rotation, ce qui permet de laisser les éléments alignés et immobiles. D'une certaine manière, en entrant dans un sous-menu *M* par l'intermédiaire d'une icône *K*, l'icône *K* serait simplement *empilée* au-dessus des icônes déja présentes sur l'axe de rotation, et le menu serait remplacé par le menu *M*. Cette idée est illustrée par la figures 4.22.

L'idée semble séductrice au premier abord. Nous l'avons donc incorporée telle quelle dans Dogme^{RV}, et proposée aux utilisateurs en test. Ce test n'était à l'origine pas destiné à évaluer cet aspect précis de la métaphore, mais la métaphore dans son ensemble, en tant que technique de contrôle d'application dans une application relativement complexe. Malheureusement, la représentation se révéla insatisfaisante. Pour ainsi dire tous les utilisateurs se perdaient régulièrement dans les menus et sous-menus, et ne remarquaient pas d'eux-mêmes la raison d'être de cet empilement de boites au centre de l'anneau. Et même une fois expliqué, l'empilement ne leur semblait d'aucun secours.

Tous les utilisateurs s'accordaient sur le fait qu'ils se sentaient totalement perdus dans les menus. Souvent, leur seul recours était de recommencer au sommet de la hiérarchie et de descendre jusqu'à l'élément qu'ils cherchaient, en espérant ne pas se tromper dans la navigation. On peut évidemment imputer une partie de ceci à la conception des menus eux-mêmes, qui n'est peut être pas idéale. Une autre partie peut certainement être imputée à la non expérience des utilisateurs sur le logiciel. Néanmoins, il nous a semblé inquiétant qu'aucun d'eux ne se soit servi spontanément du chemin inverse de navigation pour s'aider à ne pas se perdre.

Nous avons donc revu la représentation de cette information de navigation. Notre réflexion a en partie été dirigée par les remarques et suggestions faites par les

On peut voir le chemin de navigation emprunté pour aboutir à cette page à deux endroits, cerclés de rouge.

FIG. 4.21 – Une page du catalogue de sites Yahoo™.

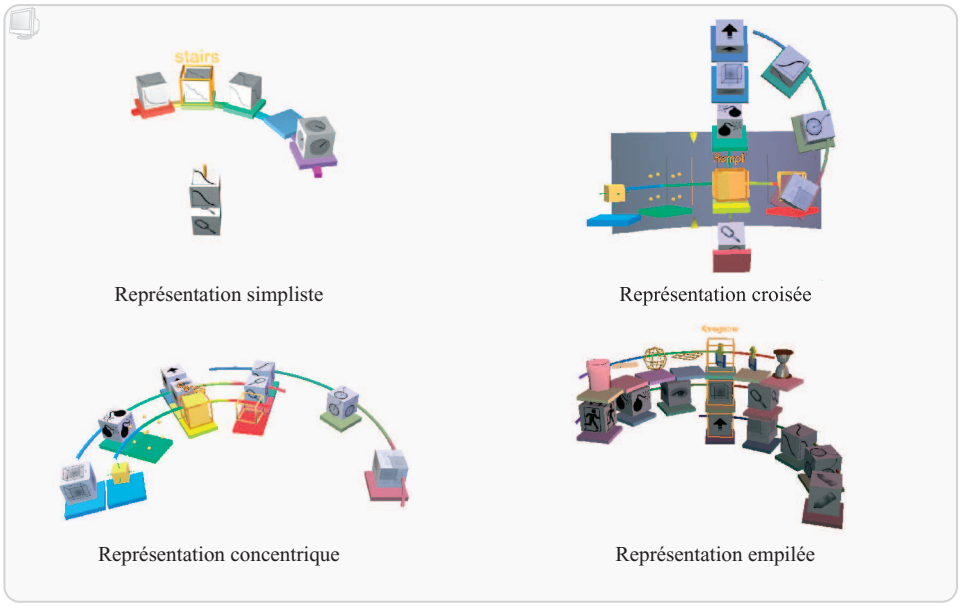


FIG. 4.22 – Différentes représentations de la hiérarchie.

utilisateurs. Nous avons également testé deux autres représentations, plus simples, qui à l'usage se sont révélées plus pratiques. Les différentes représentations sont décrites dans les paragraphes suivants. Les tests de validations effectués sur ces représentations sont présentés dans la section suivante.

Représentation croisée

Cette représentation était suggérée par quelques utilisateurs. Elle consiste à représenter un niveau de menu orthogonalement au niveau précédent. Par exemple, si un menu est affiché horizontalement, alors le prochain sous-menu sera représenté verticalement, puis à nouveau horizontalement, et ainsi de suite. La manipulation se trouve tout logiquement adaptée à cet affichage : les menus *horizontaux* sont manipulés par rotation dans le plan horizontal, alors que les menus *verticaux* sont manipulés par rotation dans le plan vertical. Le geste de fermeture de menu reste orthogonal au plan de manipulation. Nous reviendrons sur ce point dans la section 4.7. Visuellement, à l'ouverture d'un sous-menu, un menu orthogonal appairait, positionné dans l'axe de l'élément qui a permis de l'ouvrir. L'intersection des deux anneaux permet donc de retrouver son chemin de navigation. Afin de rendre ces éléments plus visibles, ils restent également encadrés de la boite orange, comme l'est l'élément actif du menu courant.

L'intérêt présumé de cette représentation est de tirer profit d'une deuxième dimension (visuellement et éventuellement en manipulation), pour l'instant laissée vacante par la métaphore.

Représentation concentrique

Cette représentation, visuellement moins complexe que la précédente, propose de placer chaque sous-menu à l'intérieur de son menu père, dans un même plan. En réalité, le menu père s'agrandit de manière à recevoir le menu fils. Cette représentation ne requiert aucun changement dans la manipulation, puisque chaque nouveau menu est affiché exactement comme l'était l'ancien. D'un point de vue visuel, on se retrouve donc avec un ou plusieurs anneaux concentriques, dont l'anneau central est l'anneau manipulé. Les anneaux conservent leur angle de rotation lorsqu'un nouvel anneau est ouvert, ce qui permet de lire le chemin de navigation en regardant l'alignement d'éléments derrière l'élément actif du menu courant.

Au fur et à mesure qu'un menu est repoussé vers l'extérieur, sa taille augmente, puisque son rayon augmente. Les éléments sont donc de plus en plus éloignés les uns des autres. Il est également possible de conserver un espacement identique des éléments en fixant le périmètre linéaire de l'anneau.

L'intérêt présumé de cette représentation est de proposer une représentation très lisible du chemin de navigation, puisqu'il réside directement dans le champ de vision de l'utilisateur. La manipulation n'est pas alourdie par la hiérarchie.

Représentation empilée

Cette représentation se base sur un empilement, mais de menus complets cette fois. Ceux-ci forment un empilement vertical de menus, dont le sommet est le menu actif. Lorsque l'utilisateur pénètre dans un sous-menu, le menu qu'il manipule descend d'un cran, en conservant sa rotation. Le chemin de navigation est formé par tous les éléments directement sous l'élément actif. Il réside donc en permanence dans le champ de vision de l'utilisateur.

Nous avons choisi d'empiler les éléments vers le bas vu la manière dont les utilisateurs manipulaient d'eux-mêmes la métaphore, c'est à dire sous leur regard. Cette constatation est probablement liée au dispositif et au périphérique d'entrée. Il serait parfaitement envisageable d'opter pour un empilement vers le haut dans un environnement différent. Dans notre configuration, conserver le menu courant au dessus de la pile de menus permet de le conserver visible, au prix de cacher légèrement les menus parents.

L'intérêt présumé de cette méthode est d'être la plus compacte des trois, tout en ayant une représentation très visible du chemin de navigation. D'un point de vue cognitif également, cette méthode est celle qui demande le moins d'adaptation de la part de l'utilisateur lors de l'ouverture puisque le menu père conserve sa taille et son apparence.

4.8.3 Validation

Afin de déterminer quelle représentation de la hiérarchie de menus était préférable, nous les avons soumises à des utilisateurs. Le principe du test était d'effectuer une sélection dirigée dans une hiérarchie de menus. Les menus étaient composés chacun de 7 éléments, numérotés de 1 à 7. Chacun d'eux ouvrait un autre sous-menu, identique, ceci jusqu'à une profondeur de quatre menus. La tâche de sélection consistait en la sélection d'un nombre à quatre chiffres. Le premier chiffre référait au menu de niveau 1, le second chiffre au menu de niveau 2, etc. Par exemple, si le chiffre à sélectionnér était "4625", l'utilisateur devait entrer dans le sous-menu numéro 4, puis le 6, le 2 et enfin le 5. La figure 4.23 donne un exemple de menu empilé durant le test de performance.

Nous proposons d'évaluer chaque représentation selon trois critères.

Sélection directe (*D*) : Nous dirons qu'un utilisateur fait une sélection directe lorsqu'il parcourt le chemin optimal dans la hiérarchie de menus jusqu'à l'élément final. Ce type de sélection est le plus souhaitable, puisqu'il évite toute perte de temps et de relation cognitive avec la tâche à accomplir. Il est donc souhaitable de maximiser ce type de sélection.

Sélection indirecte (*I*) : Nous dirons qu'un utilisateur fait une sélection indirecte lorsqu'il parvient à sélectionner le bon élément, mais en empruntant un chemin non optimal. Cela se traduit par l'entrée –et la sortie– non nécessaires dans un ou plusieurs sous-menus. Cependant, l'utilisateur s'en est rendu

compte et est revenu en arrière. Ce type de sélections, bien que moins optimal qu'une *sélection directe* ne produit pas pour autant d'erreur. On peut noter également que cette grandeur est reliée à la présence et la clarté de présentation du chemin de navigation, qui permet de se rendre compte de son erreur.

Sélection erronée (E) : Nous dirons qu'un utilisateur fait une sélection erronée lorsqu'il ne parvient pas à sélectionner le bon élément, qu'il soit ou non entré dans un ou plusieurs sous-menus non nécessaires. Ce type de sélection représente les erreurs franches, soit de manipulation lorsque l'utilisateur *dérape* sur l'élément qu'il veut sélectionner, soit lorsqu'il se trompe de menu. Il est donc souhaitable de minimiser cette valeur.

Afin d'ordonner les différentes représentations, nous proposons de tenir compte des trois grandeurs précédemment citées, selon certains coefficients. Nous cherchons donc à trouver une représentation qui fasse que :

$$\alpha D + \beta I - \gamma E \quad \text{soit maximal}$$

avec α, β et γ des coefficients de pondération selon l'importance relative assignée à chaque critère. Nos avons utilisé des coefficients de $\alpha = 2, \beta = 1 et \gamma = 3$. (notons que $E + I + D$ représente toutes les sélections faites par l'utilisateur, donc si $\alpha D + \beta I$ est maximal, E sera nécessairement minimal).

Les résultats du test sont montrés dans le graphique 4.24 et le tableau 4.4. On peut remarquer que l'une des représentations est réellement moins pratique que les autres : la représentation *croisée*. En effet, à la fois les temps de sélections sont nettement supérieurs, ainsi que le nombre d'erreurs de sélection. Si l'on analyse de plus près, on remarque que les niveaux impairs ont des temps de sélection plus élevés, or ce sont eux qui étaient affichés et manipulés verticalement. On peut donc noter que la discontinuité cognitive que la métaphore impose aux utilisateurs impose un effort de réflexion important, qui ralentit donc les performances. Nous avons essayé de laisser la manipulation toujours dans le plan horizontal (malgré l'affichage vertical), ce qui donnait des résultats encore moins bons. Au cours des tests, il était courant que les utilisateurs ne se souviennent plus dans quel plan tourner le poignet pour bouger l'anneau. Il faut noter également que les temps moyens de sélection d'un élément augmentent pour chaque sous-menu supplémentaire. Ceci est d'autant plus étonnant que c'est cette version qu'ils avaient eux-mêmes proposé en tant que solution.

Les deux autres représentations sont, comme on peut le voir, relativement équivalente en terme de précision et de performance. La version *empilée* est très légèrement plus efficace, quoique cette affirmation peut être discutée puisque l'analyse de variance donne une probabilité d'hypothèse nulle de e $p = 0.064$. Nous avons tout de même retenu la représentation empilée, pour son plus faible espace occupé sur l'écran et sa meilleure cohérence. En effet, des trois représentations, c'est elle la plus compacte, ce qui à notre avis a son importance, puisque cela permet de réduire d'autant les risques d'occlusion du menu par le reste de la scène. En termes de cohérence également, c'est elle qui modifie le moins l'apparence des menus

FIG. 4.23 – Menu hiérarchique durant le test de performances.

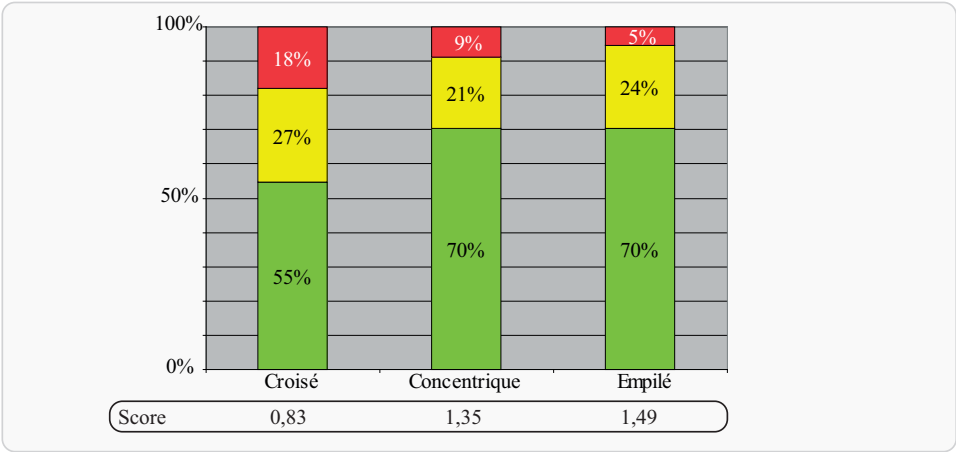


FIG. 4.24 – Proportions de sélections bonnes, indirectes et erronées

	Niveau 0	Niveau 1	Niveau 2	Niveau 3
Croisé	1.59	2.05	1.97	2.22
Concentrique	1.61	1.50	1.52	1.48
Empilé	1.51	1.37	1.39	1.36

TAB. 4.4 – Temps de sélection dans une hiérarchie

et sous-menus. D'une manière générale également, les utilisateurs préféraient –de manière informelle– cette version par rapport aux deux autres.

4.9 Conclusion

Nous avons vu dans ce chapitre l'élaboration d'une métaphore de menu adaptée aux environnements virtuels. Celle-ci permet aussi bien à des utilisateurs novices qu'experts de manipuler efficacement une application. Les tests que nous avons effectués montrent que d'une manière générale, les utilisateurs apprécient la manipulation proposée, et trouvent celle-ci facile à comprendre. L'intégration dans le prototype Dogme^{RV} montre que la technique est viable. Elle a également été reprise avec succès dans une application de simulation de planté d'aiguilles pour la préparation d'opérations par radiofréquences.

Cette métaphore est destinée principalement à des utilisateurs novices à intermédiaires. Les utilisateurs experts la trouveront peut être trop visuelle à leur gout. Elle est en effet peu adaptée à une manipulation en aveugle. Pour ce type d'utilisations, une métaphore différente est à préconiser, comme le Command and Control Cube, réfléchie dès le départ pour ce mode d'utilisation. En fait, les deux techniques semblent se compléter : le Spin Menu est rapidement accessible aux utilisateurs débutants ou en phase d'apprentissage, qui ne recherchent pas la rapidité mais l'aisance et la facilité de manipulation. Une fois l'application bien connue, ou pour des tâches répétitives, une manipulation de type «raccourci» est plus adaptée ; le Command and Control Cube se place en excellente position pour celà. L'ennui avec cette approche est qu'elle repose sur deux paradigmes dont la manipulation est totalement différente, et obligerait les utilisateurs à faire un effort d'adaptation plus ou moins conséquent. Lors du choix d'une technique de menu, il faut donc tenir compte du public auquel est destiné l'application : pour une application dont les utilisateurs changent régulièrement, n'ayant pas le temps de devenir «experts», une technique «visuelle» comme le Spin Menu est à préconiser. Au contraire, pour une application sur laquelle les utilisateurs passeront un temps considérable et auront ainsi beaucoup d'entrainement, une technique de type «raccourci» est plus adaptée.

L'un des aspects intéressants de la métaphore réside dans la possibilité de la manipuler avec la main non dominante. Un test synthétique montre que cette hypothèse est exacte. Cependant, il faut à présent vérifier ceci dans une application réelle, où la capacité cognitive de l'utilisateur n'est pas dédiée à la tâche de manipulation du menu, mais attribuée à une véritable tâche. Car n'oublions pas que le contrôle d'application a pour ultime but de déranger le moins possible l'utilisateur : la meilleure métaphore serait celle que l'on utilise sans s'en rendre compte...

Chapitre 5

Implantation

5.1 Introduction

Afin de mettre au point le prototype Dogme^{RV} donc nous avons déjà abondamment parlé, ainsi que pour me permettre d'expérimenter diverses techniques d'interaction, j'ai regroupé un large nombre de fonctionnalités au sein d'une librairie d'interaction de bas niveau, la libSelect. Une seconde librairie a également été développée afin de pouvoir utiliser un environnement expérimental, composé d'un PC classique équipé d'une carte graphique à deux sorties vidéos, chacune connectée sur l'un des projecteurs de notre dispositif. N'ayant aucune licence de notre environnement de développement historique, la CAVElib[1], pour un PC Windows, j'ai donc réécrit la gestion de la projection et de la stéréo, que j'ai rendue disponible pour la réutilisation sous forme d'une librairie, la libCube. Ces deux librairies forment la base sur laquelle est construite Dogme^{RV} : l'abstraction matérielle des écrans et des périphériques d'entrées est faite par la libCube, reposant elle-même sur OpenGL et TrackD, tandis que l'ensemble de l'interaction est prise en charge par la libSelect. La figure 5.1 donne une illustration de l'imbrication de ces librairies.

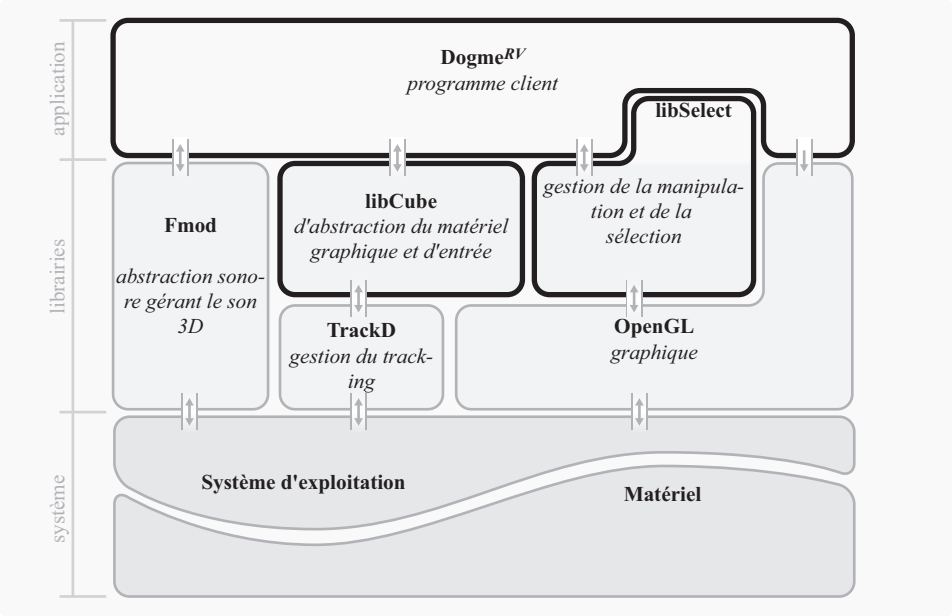
Nous allons dans ce chapitre brosser un petit portrait de ces deux librairies. Ce chapitre ne se veut pas particulièrement technique et vise plutôt à être une présentation d'ordre général de ces librairies.

5.2 Librairie *Select*

5.2.1 Introduction

A l'origine, la libSelect était destinée à faciliter la création d'interfaces graphiques 3D, basées principalement sur des fenêtres et des boutons. Elle intégrait

[1] La CAVELib est une librairie de bas niveau offrant une abstraction de la disposition des écrans et des périphériques d'entrées, par un mécanisme de fonctions de rappels (ang. *callbacks*) très comparables à la librairie *glut*.



Les parties que j'ai écrites sont entourées d'un trait sombre plus épais.

FIG. 5.1 – Imbrication des librairies

également un mécanisme de désignation accéléré matériellement par OpenGL. Elle a ensuite évolué pour devenir plus généraliste et permet actuellement de créer et de manipuler tout type d'objet. Son nom quelque peu étrange provient du nom de la classe C++ de base de tous les objets, `Select0`, qui porte ce nom car elle est la base de tous les objets que l'on peut *sélectionner*.

5.2.2 Philosophie

Toute la librairie repose sur le mécanisme du passage de messages. La figure 5.2 illustre le schéma global de fonctionnement de la librairie. Chaque évènement qui peut se produire est signalé à un ou plusieurs objets par un message représenté par un entier, éventuellement complété par une très petite information. Les objets qui reçoivent le message réagissent alors à leur manière, ce qui leur confère un certain comportement. On peut distinguer deux types de messages, qui sont les messages *système* et les messages *utilisateur*.

Messages système

A la réception d'un message système, tout objet est présumé adopter un comportement prédéfini, qu'il peut légèrement adapter, mais tout en gardant la sémantique associée au message. Par exemple, sur un évènement indiquant un déplace-

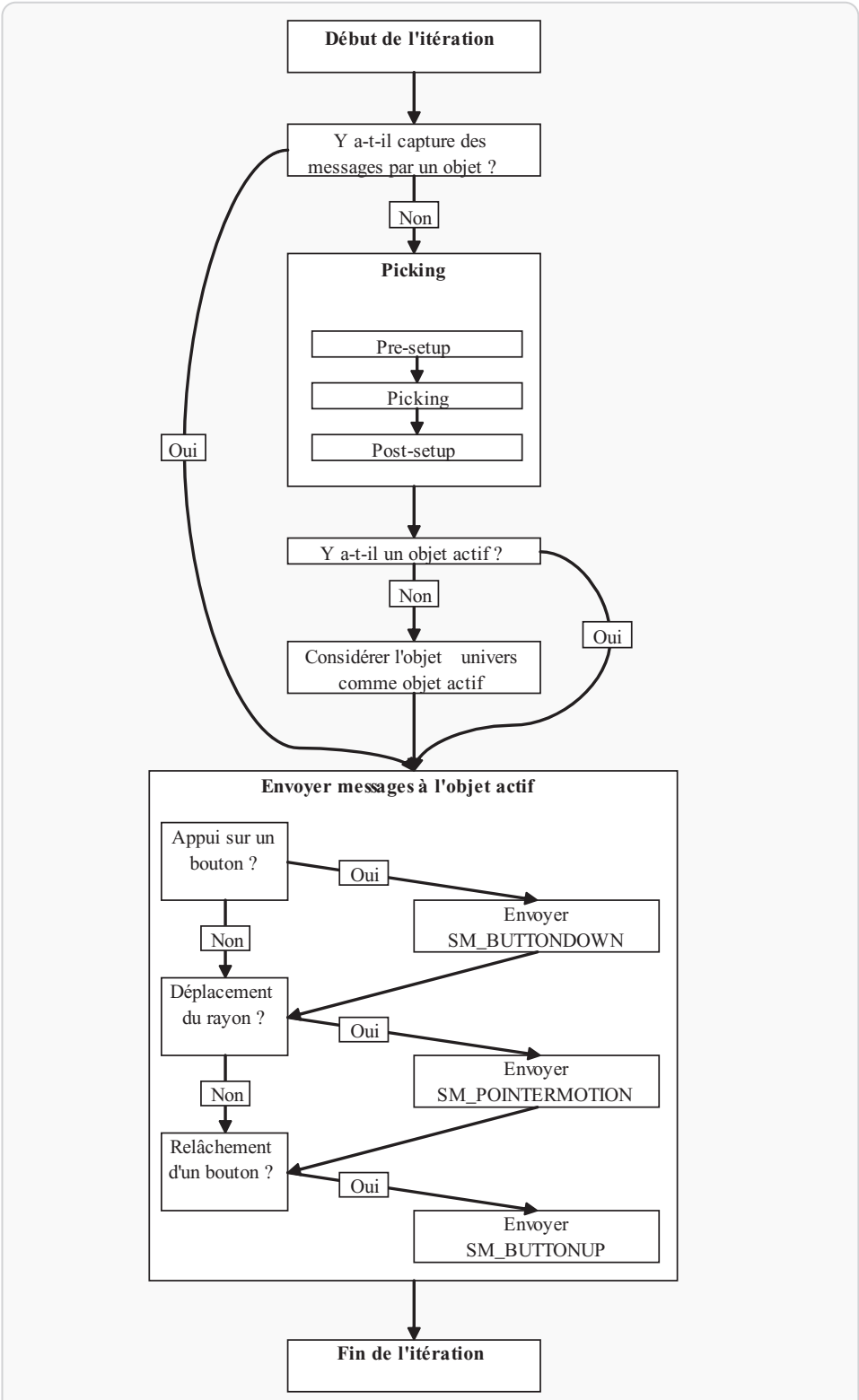


FIG. 5.2 – Schéma de fonctionnement de la librairie.

ment de l'objet, chaque objet est présumé se placer à la position indiquée. Il est néanmoins possible, dans certains cas, de ne pas implanter ce comportement par défaut, par exemple pour créer des objets qui ne sont pas déplaçables. Les messages systèmes sont relativement peu nombreux, et forment la couche de base de la librairie. Leur comportement par défaut n'est pas à réécrire pour chaque nouvel objet créé, seuls les comportements modifiés doivent être réécrits.

Messages utilisateur

Au contraire des messages systèmes, les messages utilisateurs n'ont aucun comportement associé par défaut. Chaque objet peut donc s'en servir pour implanter des comportements non prévus à l'origine, ou étendre des comportements existants en leur offrant plus de souplesse. Par exemple, en imaginant un objet *fenêtre*, le fait de minimiser la taille de la fenêtre serait implantée dans un message utilisateur. Bien entendu, il faut éviter d'envoyer des messages utilisateurs d'un objet A à un objet B, puisque cet objet B pourra avoir un comportement totalement différent de A vis-à-vis du même message utilisateur.

Architecture

D'un point de vue architectural, la librairie est composée d'un objet C++ servant de base (par héritage) à tous les objets gérés par la librairie. Un certain nombre de fonctionnalités communes sont d'ores et déjà implantées dans cet objet, afin que tout objet futur puisse en bénéficier, comme l'implantation des messages systèmes. Chaque objet peut ensuite être instancié autant de fois que nécessaire, en le référençant simplement par un nom symbolique que l'objet se donne à lui-même. Par exemple, pour créer un objet de type *bouton poussoir*, il suffit d'instancier dynamiquement un objet nommé *Select_Button*. Ce nom symbolique représente le nom de la famille d'objets ayant tous le même comportement vis-à-vis des évènements. Le diagramme 5.3 illustre l'ensemble des familles d'objets que propose la librairie. Chacune de ces classes peut être étendue par l'utilisateur via un mécanisme classique d'héritage.

Désignation et manipulation

Les objets sont principalement manipulés par des messages. Une large majorité d'entre eux proviennent de l'utilisateur, et plus particulièrement de son périphérique d'entrée. La librairie nécessite de façon intrinsèque une méthode de désignation des objets, afin de savoir à quel objet envoyer les messages générés par l'interaction. La manipulation est laissée à la discrétion de chaque objet, et sera implantée à sa guise en attachant le comportement adéquat à chaque message issu de l'interaction avec l'utilisateur. La désignation quant à elle est un service proposé par la librairie. Elle repose sur l'utilisation d'une fonctionnalité OpenGL permettant de savoir quels objets sont affichés à l'écran : le *picking*. Cette technique

permet, moyennant un nommage judicieux des différents triangles que compose la scène, de déterminer l'ensemble des primitives qui sont dessinées à l'écran. En ajustant correctement les matrices de projection, on peut donc implanter avec le *picking* un rayon laser virtuel totalement pris en charge par le matériel. Pour cela, il suffit d'imaginer que le périphérique d'interaction dispose d'un oeil, qui *voit* uniquement ce qui doit être désigné, c'est à dire qu'il a un champ de vision très étroit, et orienté exactement comme le périphérique. Le mécanisme de *picking* fonctionne exactement comme l'affichage, sauf qu'au lieu de générer des pixels, chaque primitive qui serait affichée est enregistrée dans un espace mémoire avec quelques informations supplémentaires. On peut ainsi savoir ce que *voit* le laser virtuel, et en déduire l'objet le plus proche de tous. Cet objet particulier recevra alors tous les messages d'interaction, comme l'information que le laser le désigne actuellement, les évènements de pression et de relâchement des boutons, etc. La figure 5.4 illustre ce principe. Le principal avantage de ceci, par rapport aux classiques méthodes de lancer de rayon, est qu'il peut fonctionner sur n'importe quel type d'objets que l'on sait afficher, même si ce n'est pas la librairie qui en fait l'affichage. Il garanti également une parfaite cohérence entre les données vues par l'utilisateur et les données utilisées pour la désignation. De plus, étant pris en charge par le matériel graphique, la technique est extrêmement rapide.

Partant de cette technique, il est possible de proposer diverses variations. Par exemple, le *picking* peut être soit perspective, soit isométrique. Dans le premier cas, les objets lointains seront plus aisés à désigner, au risque de sélectionner plus aisément un mauvais objet. Le second cas est exactement l'opposé. On peut également restreindre en profondeur la zone sur laquelle le *picking* opère, et par exemple effectuer une sélection de "tout ce qui est à l'intérieur d'un parallélépipède rectangle non isotrope". Nous verrons plus précisément ces techniques dans les sections suivantes.

5.2.3 Familles d'objets

Dans cette section, nous allons décrire quelques classes d'objets importantes et largement réutilisées dans les quelques programmes qui utilisent la librairie. Ces différents objets sont illustrés dans la figure 5.7.

Élement manipulable

Nom de code : `Select_Draggable`

La plupart des éléments de la librairie et des programmes sont manipulables directement par l'utilisateur, par l'intermédiaire de la métaphore de manipulation. Ce comportement est proposé par un objet, dont bon nombre héritent, comme les fenêtres 3D. Il propose simplement le mécanisme de base de la manipulation au rayon laser. Lorsque le laser désigne l'objet, et que l'utilisateur appuie sur un bouton, l'objet *s'accroche* au rayon laser. Tout mouvement du rayon laser est répercuté sur l'objet, de sorte qu'il n'y ait aucun mouvement relatif entre l'objet et le rayon.

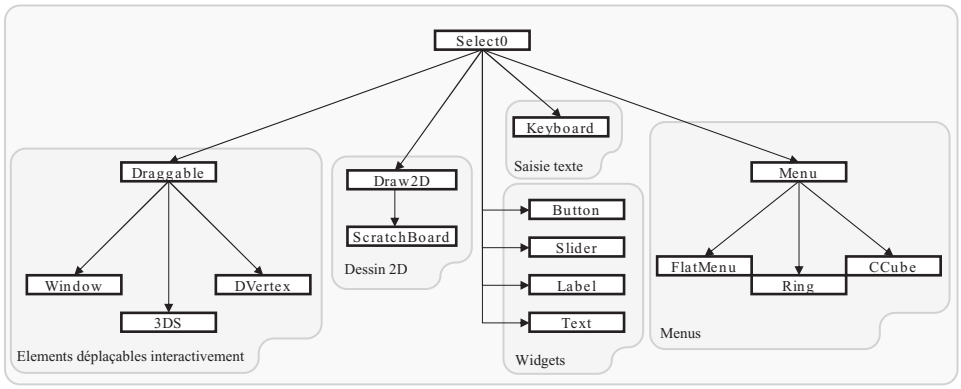
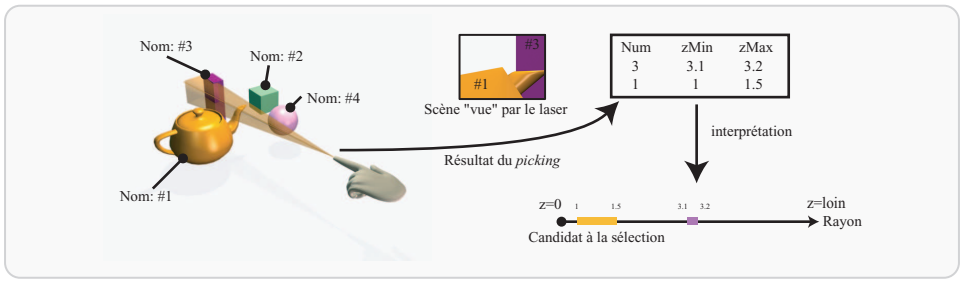


FIG. 5.3 – Diagramme des classes de la libSelect.



Lors d'une opération de *picking*, OpenGL ne produit pas de pixels mais des informations concernant chaque nom d'objet visible dans le cône de vision. Ces informations comprennent le nom de l'objet (objet étant un ou plusieurs triangles), ainsi que la coordonnée Z la plus proche et la plus lointaine de l'objet. Ces informations sont utiles pour déterminer quel objet est le plus proche du point de vue, donc de la métaphore.

FIG. 5.4 – Vision schématique du mécanisme du *picking*

Lorsque le bouton est relâché, l'objet reste dans sa nouvelle position. La manipulation est très efficace pour manipuler des objets à distance. Cependant, puisque l'on cherche à supprimer tout mouvement relatif entre l'objet et le rayon lors de la manipulation, faire faire un demi-tour à un objet sur lui-même n'est pas très pratique, étant donné que le centre de rotation se situe au centre de rotation du rayon –donc en général dans la main de l'utilisateur. Il faut donc procéder par paliers, en déposant l'objet et le reprenant pour qu'en fin de compte il soit environ dans la position souhaitée. Une telle manipulation est, comme nous l'avons déjà évoqué, *centrée sur l'utilisateur*. Pour certains objets, cette vision est bonne et correspond bien à l'attente de l'utilisateur, notamment les fenêtres 3D. Pour d'autres, comme les objets 3D, l'utilisateur a plutôt envie de les manipuler *sur place*. On adopte pour cela la vision *centrée sur l'objet*, qui place le centre de rotation au centre de l'objet. Il en découle cependant quelques difficultés à déplacer l'objet en translation, puisqu'il faut alors effectivement déplacer sa main et non juste la tourner comme on a tendance à le faire avec la métaphore du laser. Ces deux comportements sont disponibles dans ce composant, on sélectionne l'un ou l'autre de manière totalement dynamique.

Fenêtre conteneur

Nom de code : `Select_Window`

La fenêtre conteneur agit comme un cadre dans lequel on peut placer divers *widgets* 3D habituels, comme des boutons, des textes, des cases à cocher, etc. Le conteneur dispose d'une zone de titre et de quelques boutons spéciaux. La zone de titre permet de déplacer la fenêtre et de la placer librement dans l'espace de travail. Les boutons permettent de réduire la taille de la fenêtre ou de la fermer complètement. Elle est très similaire à toute fenêtre d'interface graphique moderne, excepté qu'elle dispose d'une légère épaisseur et est plaçable et manipulable en 3D. L'un des intérêts de cet objet est sa capacité à s'initialiser à partir d'un fichier de description, qui en quelques lignes définit une fenêtre avec ses widgets. La création de la fenêtre et de sa descendance se fait dans le programme en une seule ligne, et l'on peut ajuster l'interface sans avoir à recompiler le moindre code, sous certaines restrictions. La figure 5.5 donne un exemple de fichier descriptif d'interface graphique, donnant lieu à la fenêtre illustrée en 5.6.

Bouton poussoir

Nom de code : `Select_Button` et `Select_3DS`

Le bouton poussoir regroupe bon nombre de fonctionnalités communes aux boutons d'action et aux cases à cocher classiques. Il peut soit agir comme un bouton, soit comme une case à cocher –*N* choix parmi *M*, soit comme un bouton radio –1 choix parmi *M*. Son apparence peut être celle d'une boite sur laquelle est inscrite un texte ou une image, ou plus généralement un objet 3D issu d'un fichier. Ces boutons forment une base pour toute interface graphique. Utilisés judicieusement,

Classe d'élément à créer	Identifiant unique de l'objet	*Flags* permettant de paramétrer le comportement ou l'apparence de l'objet	Texte ou arguments de création	Position 3D initiale	Taille 3D initiale

```
#
# Proprietes des contraintes
#

@symbols   "symbols.h"

Select_Window   IDD_CONTRAINTE_PROPS SS_GLOBALROTATE                                  "Proprietes"            -0.5  0.65  0  0.42 0.52 0.01
@children
 #
 # Volumes d'influence (sphere, boite, etc.)
 #
 Select_Label   0                    SS_VISIBLE                                        "Volume d'influence"    0.02 -0.02 0  0    0    0
 Select_3dsMesh IDC_VOLUME_SPHERE    SS_VISIBLE|SBS_RADIOBUTTON|SMS_SMOOTH|SS_GROUPSTART "icons/vol_sphere.3ds" 0.02 -0.06 0  0.05 0.05 0.05
 Select_3dsMesh IDC_VOLUME_BOX       SS_VISIBLE|SBS_RADIOBUTTON                        "icons/vol_box.3ds"     0.09 -0.06 0  0.05 0.05 0.05
 Select_3dsMesh IDC_VOLUME_VOXL      SS_VISIBLE|SBS_RADIOBUTTON                        "icons/vol_voxel.3ds"   0.16 -0.06 0  0.05 0.05 0.05
 Select_3dsMesh IDC_DRAW_VOXL        SS_VISIBLE|SBS_CHECKBOX                           "icons/draw.3ds"        0.23 -0.06 0  0.05 0.05 0.05
 Select_3dsMesh IDC_ERASE_VOXL       SS_VISIBLE|SBS_CHECKBOX|SMS_SMOOTH                "icons/eraser.3ds"      0.30 -0.06 0  0.05 0.05 0.05

 #
 # Fonctions de transfert distance->influence (cos^n, etc.)
 #
 Select_Label   0                    SS_VISIBLE                                        "Fonction de transfert" 0.02 -0.14 0  0    0    0
 Select_3dsMesh IDC_INFLUENCE_DOOR   SS_VISIBLE|SBS_RADIOBUTTON|SS_GROUPSTART         "icons/inf_door.3ds"    0.02 -0.18 0  0.10 0.10 0.10
 Select_3dsMesh IDC_INFLUENCE_COSN   SS_VISIBLE|SBS_RADIOBUTTON                        "icons/inf_cosn.3ds"    0.14 -0.18 0  0.10 0.10 0.10
 Select_3dsMesh IDC_INFLUENCE_XN     SS_VISIBLE|SBS_RADIOBUTTON                        "icons/inf_xn.3ds"      0.26 -0.18 0  0.10 0.10 0.10
 Select_3dsMesh IDC_INFLUENCE_STEPS  SS_VISIBLE|SBS_RADIOBUTTON                        "icons/inf_steps.3ds"   0.38 -0.18 0  0.10 0.10 0.10
 Select_3dsMesh IDC_INFLUENCE_USER   SS_VISIBLE|SBS_RADIOBUTTON                        "icons/inf_user.3ds"    0.50 -0.18 0  0.10 0.10 0.10

 #
 # Slider pour les fonctions de transfert paramétriques (cos^n et x^n)
 #
 Select_Label   0                    SS_VISIBLE                                        "Parametre"             0.02 -0.31 0  0    0    0
 Select_Label   IDC_INFLUENCE_PARAML SS_VISIBLE                                        "1.00"                  0.35 -0.31 0  0    0    0
 Select_Range   IDC_INFLUENCE_PARAM  SS_VISIBLE                                        "1/400/100"             0.02 -0.35 0  0.40 0.05 0.05

 #
 # Modifier le chemin de contrainte
 #
 Select_Label   0                    SS_VISIBLE                                        "Contrainte"            0.02 -0.42 0  0    0    0
 Select_3dsMesh IDC_RECORD_DPATH     SS_VISIBLE|SBS_PUSHBUTTON                         "icons/draw.3ds"        0.02 -0.48 0  0.05 0.05 0.05
 Select_3dsMesh IDC_MAGNET_MODE      SS_VISIBLE|SBS_PUSHBUTTON                         "icons/magnet.3ds"      0.12 -0.48 0  0.05 0.05 0.05

@endchildren
```

FIG. 5.5 – Exemple de fichier descriptif d'interface graphique

ils peuvent donner naissance à des interfaces plus inattendues, comme des pseudo menus déroulants

Zone de dessin 2D

Nom de code : `Select_Draw2D`
La zone de dessin 2D simule le comportement d'une feuille de papier. Le rayon laser permet d'y laisser une trace, que l'on peut utiliser pour fournir le comportement final de l'objet. Par défaut, la trace est simplement enregistrée dans une texture et affichée comme telle, ce qui donne l'impression de dessiner sur l'objet. Bien entendu le composant est plutôt destiné à servir de base à d'autres composants, comme la saisie de fonctions d'extrusion dans Dogme^{RV}.

Menus

Nom de code : `Select_Ring`, `Select_CCube` et `Select_FlatMenu`
La famille des composants de type *menu* propose des fonctionnalités différentes des autres composants. Ils fournissent à une hiérarchie de menu sa représentation graphique. Les menus eux-mêmes sont extraits d'un autre composant, qui lui-même s'initialise à partir d'un ou plusieurs fichiers externes. A chaque représentation graphique d'un menu est associée une manipulation précise –le spin menu se manipule par rotation, le C^3 se manipule par déplacement, etc.. L'idée est de pouvoir changer de métaphore de menu à la volée, sans avoir à modifier le programme en profondeur.

5.2.4 Techniques de désignation

Laser

La métaphore par défaut de la librairie est le laser virtuel, proche de celui décrit dans [Min95b]. Comme indiqué précédemment, l'ensemble de la métaphore est implantée en utilisant les mecanismes OpenGL standards, et est donc très performante. De ce fait, même si l'on parle généralement d'un cône de sélection, on travaille en réalité avec une pyramide ou un parallélépipède de sélection, puisque l'on repose sur OpenGL pour qui les écrans sont nécessairement rectangulaires. Il est possible d'ajuster un grand nombre de paramètres, comme l'ouverture (*aperture*) de la pyramide de sélection, sa profondeur minimale et maximale, son rapport –pour le transformer en rectangle, l'utilisation ou non de la perspective dans le processus de sélection, etc. Ceci en fait une métaphore relativement polyvalente. Nous avons trouvé cette métaphore globalement très satisfaisante, et surtout très intuitive à utiliser et à manipuler.

Outre le laser standard, on peut bien entendu utiliser toutes les variantes proposées par la littérature, comme le laser dirigé par le regard, la sélection par *image plane* [PFC+97], etc. en jouant sur l'origine, la direction et la taille de la pyramide de sélection. La figure 5.8 illustre quelques techniques, détaillées ci dessous.

Laser à mémoire

Le laser à mémoire, inventé par Ludovic Sternberger [Ste03] est une métaphore de rayon laser dont le rayon peut être infléchi par l'utilisateur. Ceci lui permet d'atteindre des objets partiellement occultés plus aisément (en théorie). L'idée est de «ralentir» le laser, et de pouvoir influencer sur sa trajectoire au cours du temps : l'utilisateur tourne sa main pour courber le rayon et ainsi éviter les obstacles. Le premier objet rencontré arrête le processus, et l'on embraye sur une manipulation *laser* classique. La même métaphore a été implantée dans la libSelect en utilisant le *picking*. En réalité, seule l'extrémité du laser est sensible dans cette métaphore, le reste du rayon ne sert qu'à montrer visuellement la trajectoire suivie. L'affichage étant nécessairement discret –un affichage par image, le laser est en réalité composé d'un certain nombre de lignes brisées, dont celle entre l'image courante n et l'image précédente $n-1$ sera la partie active du laser. Il suffit donc de considérer une zone de *picking* orientée et positionnée selon ce segment de la ligne brisée. Si un objet intersecte cette ligne, il sera sélectionné.

Manipulation directe

Souvent sur un plan de travail virtuel on manipule des maquettes d'objets, qui sont affichées à l'échelle de l'utilisateur et du dispositif. La meilleure façon de manipuler dans ce cas est la manipulation *directe*, c'est à dire que l'utilisateur touche ou saisit ce qu'il désire sélectionner, ou tout au moins place sa main à la même position. La technique souffre cependant d'un problème d'occlusion induit par la disposition géométrique du dispositif, qui fait que la main réelle de l'utilisateur cachera toujours la scène. Néanmoins, la métaphore se prête parfaitement à être exprimée en terme de *picking*. Il suffit pour cela de considérer une zone de sélection cubique relativement restreinte au niveau de la main ou des doigts de l'utilisateur, selon le matériel dont on dispose.

5.2.5 Conclusion

La libSelect, conçue au départ pour servir mes propres besoins d'abstraction, a évolué au cours du temps en une librairie. Elle permet de simplifier considérablement la complexité de la désignation 3D par métaphore laser, ainsi que la gestion des menus. Elle a en outre été utilisée avec succès dans plusieurs autres projets, listés Figure 5.9.

5.3 Librairie *Cube*

Dogme^{RV} est un prototype fonctionnant sur un ordinateur PC standard. Il avait été développé au départ sur une Silicon Graphics, qui posait de sérieux problèmes de performance. Nous avons donc décidé de passer sur un environnement plus

récent. Malheureusement il n'y avait pas au moment du développement de l'application de plate forme logicielle de réalité virtuelle sur ce type d'environnements. Nous disposions de la CAVElib, mais exclusivement en environnements IRIX. Afin de faciliter le développement de cette application précise ainsi que les éventuelles applications futures, j'ai développé une petite librairie se chargeant d'abstraire la configuration géométrique des écrans et des périphériques d'entrées. Le même programme peut ainsi, moyennant une description correcte de l'environnement, être exécuté sur un PC standard, un workbench ou une CAVE. Cette librairie a été nommé libCube, par analogie avec la forme d'un environnement de type CAVE.

5.3.1 Possibilités

La libCube propose deux services principaux : l'abstraction de la configuration géométrique du dispositif d'affichage, et l'abstraction des périphériques de suivi de mouvement.

Par abstraction de la configuration géométrique, nous entendons le fait qu'il suffise de décrire les positions de chacun des écrans du dispositif dans le repère global, pour que la librairie s'occupe d'afficher la scène sur chaque écran, avec la matrice de projection qui convient pour l'écran considéré. Le calcul des matrices de projection pour les environnements immersifs est légèrement plus complexe que dans le cas habituel. En effet, la projection se fait non alignée sur l'axe de profondeur (en anglais, *off-axis*), comme c'est en général le cas. La pyramide de vision n'est donc plus symétrique par rapport à cet axe. Cependant, étant donné la position de l'observateur et les coordonnées de trois extrémités de l'écran, il est possible de calculer les coordonnées de la pyramide de vision de manière à obtenir une projection correcte de la scène. Les détails de ces calculs sont fournis en annexe par 149.

L'un des problèmes soulevés par l'utilisation de plusieurs écrans, en particulier dans le cas de la stéréo active, est la nécessité de synchroniser le rafraîchissement des écrans entre eux. Ceci est requis pour que les lunettes à obturation fournissent sur tous les écrans l'image du même oeil au même moment. La capacité de synchroniser plusieurs écrans est appelée *genlocking*. De plus, il faut être capable de s'attendre mutuellement à la fin du basculement de l'image. Ceci s'appelle le *framelocking*. Peu de matériels pour PC standard offrent à l'heure actuelle le support matériel pour ces deux prérequis. On peut, dans certains cas particulier, avoir recours à une solution logicielle (SoftGenlock[AGL+03]), qui permet d'émuler par un réseau la synchronisation et l'attente. Dans notre configuration, le problème était en partie évité, puisqu'une seule carte graphique à deux sortie alimentait chacun des écrans. Les deux sorties sont naturellement quasi-synchronisées entre elles étant donné qu'elles fonctionnent sur la même horloge[2].

[2]En réalité, ceci tient presque du coup de poker, le fabriquant ne donnant aucune garantie quant à cette propriété. Cette synchronisation n'est d'ailleurs pas parfaite, ce qui peut se traduire par l'apparition de bandes sombres au milieu de l'image. Dans ce cas, la seule solution est de réinitialiser l'affichage.

La librairie insère un petit étage d'abstraction entre écran physique et écran logique : chaque écran physique –ou plutôt, chaque surface d'affichage– peut être occupée par un ou plusieurs écrans logiques –normalement disjoints. Ceci permet par exemple de tenir compte de notre configuration, ou une surface d'affichage unique est divisée sur deux écrans différents, donc selon deux perspectives différentes.

La librairie offre également un étage d'indirection entre les périphériques et le programme. Elle introduit la notion de périphérique logique, un périphérique étant un *tracker* ou un *bouton*. Chaque périphérique physique est associé dans un fichier de configuration à un périphérique logique. De ce fait, on peut aisément s'adapter à un nouveau périphérique en changeant simplement l'étage d'indirection entre matériel et logiciel ; le programme n'a nullement besoin d'être modifié.

Un aspect annexe mais néanmoins intéressant que propose la librairie est le mode de simulation. Ce mode permet d'exécuter le programme sur un environnement de bureau classique, en émulant les périphériques de suivi par le clavier et la souris. Ce mode s'avère dans la pratique quasiment indispensable pour le développement d'un programme.

Enfin, le mode *replay* permet d'enregistrer l'intégralité d'une manipulation sur l'environnement virtuel, et de la rejouer par la suite autant de fois que l'on souhaite, éventuellement en mode simulation. On peut ainsi filmer une manipulation en décalé, tout en ayant les vraies actions de l'utilisateur. Pour ce faire, l'intégralité des données de suivi de mouvement sont stockées sur disque, et réutilisées par la suite, en partant du principe qu'un programme est déterministe et réagit toujours de la même manière à des stimuli identiques.

5.3.2 Interface de programmation

La libCube est écrite en C++. Elle repose largement sur la notion d'héritage : le programme doit hériter d'une classe de base afin de bénéficier des fonctionnalités de la librairie. Cette classe de base permet en réalité de forcer le nom de deux fonctions cruciales : l'affichage de la scène et le calcul entre deux images. L'utilisateur a la charge d'implanter ces fonctions comme bon lui semble. Éventuellement l'utilisateur peut assigner différentes fonctions d'affichage et/ou de calcul à chaque écran ou à un groupe d'écrans, par exemple un écran de contrôle.

Afin de gérer l'ensemble des écrans et de leurs routines d'affichage respectif, une classe de gestion est présente, qui s'occupe de synchroniser les travaux de tout le monde. Cette synchronisation se fait selon un mode multi-threadé : chaque écran physique est exécuté par un *thread* –ou processus léger– différent. La librairie s'occupe ensuite de placer les points de synchronisation entre chaque calcul et chaque affichage, de manière à garantir la cohérence de l'ensemble. Un schéma de ce fonctionnement est proposé en Figure 5.10. Un programme minimal, utilisant les fonctionnalités de la librairie, est proposé en Figure 5.11.

5.3.3 Configuration

La configuration de l'environnement matériel est décrite dans un fichier de configuration externe. Une modification à l'environnement ne demande que la mise à jour du dit fichier dans les cas favorables. Ce fichier permet, entre autres, de décrire la position des écrans physiques et logiques, de faire la relation entre périphériques physiques et logiques, etc. Un extrait de fichier de configuration est donné en Figure 5.12.

5.3.4 Conclusion

La libCube offre une interface relativement simple à utiliser en termes de programmation entre le programme et son environnement virtuel, tout en s'en éloignant un peu pour favoriser la portabilité. Enfin, la librairie a également été utilisée dans le projet *Radiofrequency* (Figure 5.9).

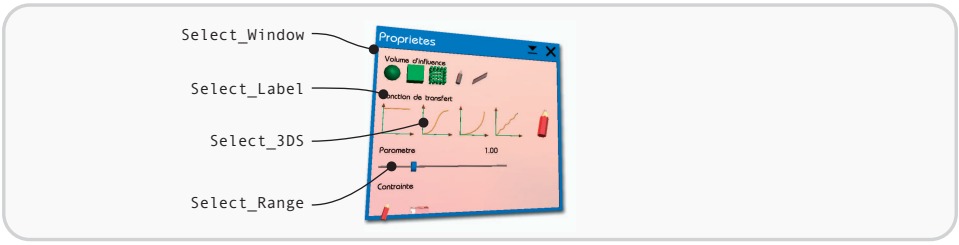
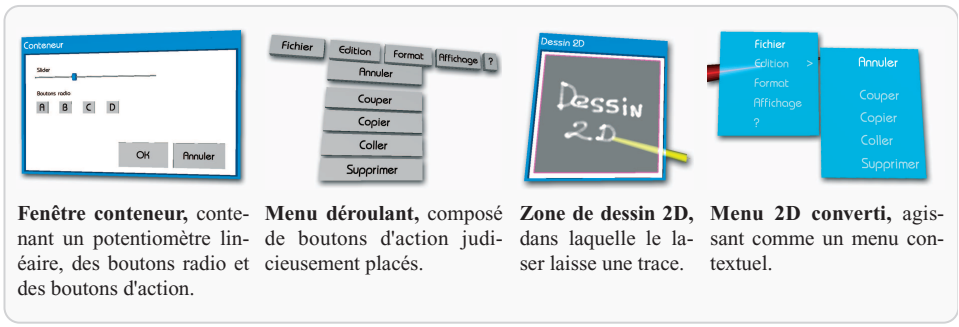


FIG. 5.6 – Fenêtre générée par le code en 5.5.



Fenêtre conteneur, contenant un potentiomètre linéaire, des boutons radio et des boutons d'action.

Menu déroulant, composé de boutons d'action judicieusement placés.

Zone de dessin 2D, dans laquelle le laser laisse une trace.

Menu 2D converti, agissant comme un menu contextuel.

FIG. 5.7 – Quelques composants proposés par la librairie.

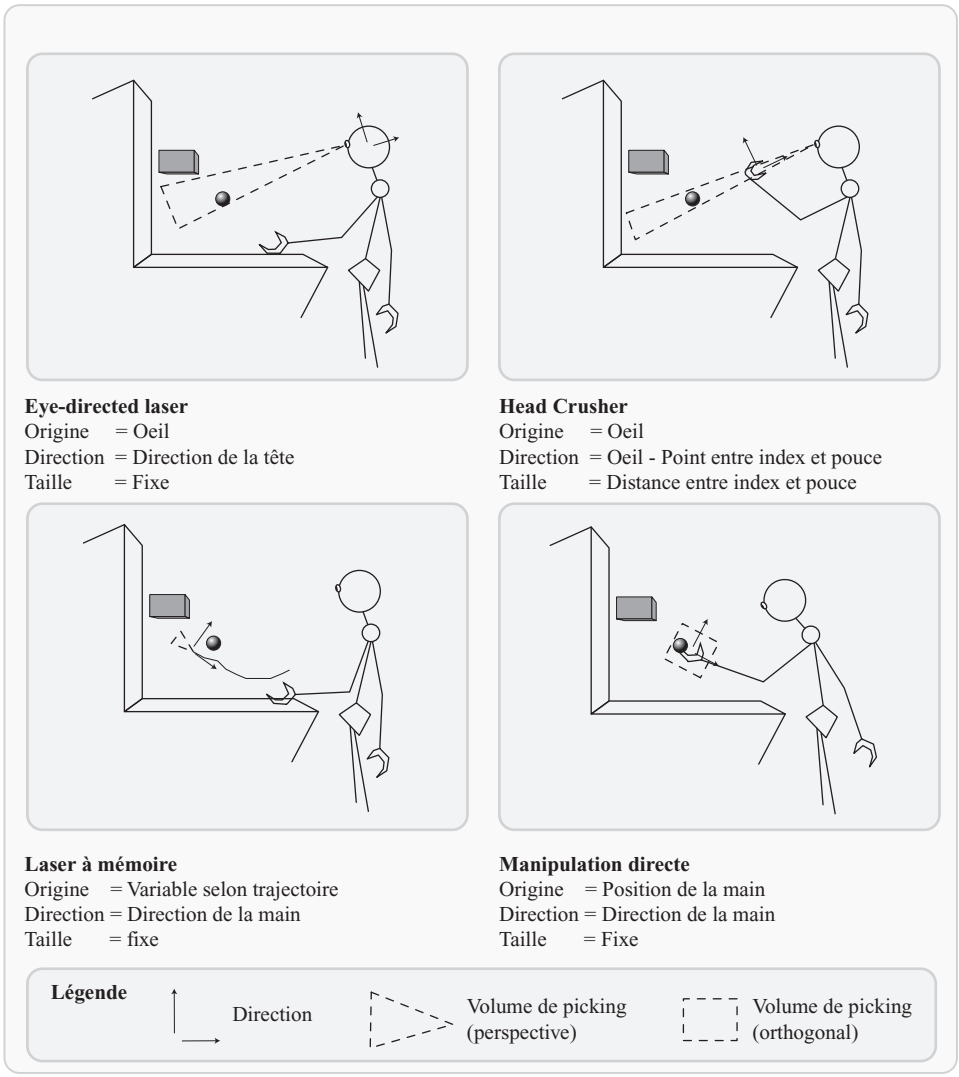
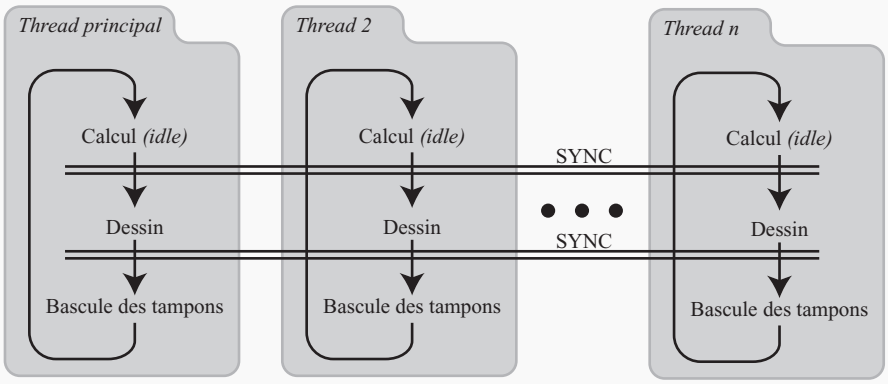


FIG. 5.8 – Différentes applications du picking

Titre	Cadre	Auteur(s)	Description	Utilisation
DogmeRV	Thèse	D. Gerber	Outil de modélisation de par déformations géométriques contraintes	Intégralité
Neuron 3D	Projet de DEA	A. Fabre	Outil de modélisation et visualisation d'un réseau de neurones en 3D	Fenêtres 3D
Coyote géomètre	Stage de DEA	A. Fabre	Outil de saisie et de résolution de figures contraintes dynamiques en 3D	Fenêtres 3D, Composants personnalisés
NiceLAB	Stage de DEA	A. Fabre, L. Sternberger, S. Besse, J. Schmid	Laboratoire d'expérimentation et de combinaison de métaphores de désignation et manipulation sur le Workbench.	Fenêtres 3D, Composants personnalisés
Radio-frequency	Stage de DUT	Y. Moalic	Adaptation d'un outil de simulation de planté d'aiguille pour opération par radiofréquence.	Menu circulaire
Pilote géologique	Stage de DEA	M. Haefele	Outil de construction du graphe de chronologie d'un sous-sol pour la recherche pétrolière.	Fenêtres 3D

FIG. 5.9 – Les différents projets ayant utilisé la libSelect



Chaque écran physique est géré par un thread; les threads sont synchronisés entre eux en deux points. Note: les fonctions de calcul et de dessin sont optionelles et peuvent être différentes pour chaque thread.

FIG. 5.10 – Synchronisation des threads d'affichage et de calcul.

```
#include <cube.h>

class CMiniCube : public CCubeCallback
{
public:

    // Constructeur
    CMiniCube (class CCube* pCube)
        : CCubeCallback(pCube)
    {
    }

    // Fonction de dessin
    // nScreen comporte le numéro de l'écran actuellement dessiné
    // nEye indique pour quel oeil l'affichage doit être fait
    virtual void Draw (int nScreen, int nEye)
    {
    }

    // Initialisation du contexte
    // Idéal pour charger les textures
    virtual void InitGL (int nScreen)
    {
    }

    // Fonction de calcul
    virtual void Idle ()
    {
    }

};

int main(int argc, char* argv[])
{
    // Gestionnaire de programme
    CCube           cube;
    // Gestionnaire d'écran
    CCubeTest*      cubetest = new CCubeTest (&cube);

    // Charge la disposition des écrans et autres réglages
    if (!cube.loadConfig (&argc, argv))
        return false;

    // Définit les fonctions d'affichage pour les écrans
    cube.setAllCallbacks (cubetest, cubetest);

    // Lance l'application
    cube.runApp ();

    return 0;
}
```

FIG. 5.11 – Programme minimal utilisant la libCube

```
#
# Lib-Cube initialisation file
#

#
# Global settings go here
#
# Global definitions are command-line overridable, ie. to override Simulator mode, simply add
# -simulator yes|no or /simulator yes|no to the command line
#
Global {
    EyeSeparation   0.07                        # Distance between the eyes (for stereo).
                                                # Units must be coherent with screen positions.

    Units           m                           # Units names. The library's unitless, but can print
                                                # a unit symbol next to values (for human readability)

    Verbosity       1                           # Verbosity of the library
                                                # The higher the number, the more messages you get.
                                                # Default is 0, for "only erratic stuff"

    CubeOrigin      <0,1.20,-0.655>             # Logical origin (opengl's origin) in world coordinates
                                                # Must be coherent with screens !

    Near            0.01                        # Near clipping plane distance
    Far             8.00                        # Far clipping plane distance

    HeadSensor      0                           # Physical tracker where the head tracker is connected to
    RightHandSensor 1                           # Physical tracker where the right hand tracker is connected to
    LeftHandSensor  2                           # Physical tracker where the left hand tracker is connected to

    Simulator       Yes                         # If on, run in simulation mode
                                                # (ie. one display only, display oultines of screens)

}

Simulator {
    SimViewerOffset     0                       # Offset for third person view

    # Position and direction of default simulator view
    SimViewerPos        <1.224, 0.663001, 1.582>
    SimViewerDir        <-0.558164, -0.146259, -0.816738>
    SimFOV              45

    WindowSize          800x573
    SimDisplayWalls     false
}

#
# Screen definitions go here
#
# You can have as many screens as you want.
#
Screen "Front" {                                # Give the screen a little name (case insensitive)


    DisplayMode         Stereo                  # Do we want to display in Mono or Stereo
                                                # (if stereo is not available, defaults to stereo with a warning)

    SwapStereo          No                      # If true, right eye is displayed on left buffer and vice-versa

    XDisplay            ":0.0"                  # Name of an X server to open display on (unix only, ignored on win32)

    Viewport            <0,0.5,1,0.5>           # Viewport to cover on this display: <x,y,w,h> (default: <0,0,1,1>)
                                                # Coordinates are relative, full viewport is assumed "1x1".


    BottomLeft          <-0.90, 1.20, -1.31>    # Physical coordinates of the screen.
                                                # We need only three coordinates to determine the fourth
    UpLeft              <-0.90, 2.55, -1.31>
    BottomRight         < 0.90, 1.20, -1.31>

    MatteMask           xxxxx...                # Matte caches to display around the screen (in video mode)

}


#
# Antitrax tracking system config
#
Antitrax {
    Connect             Simulator               # Where to get data from (Simulator,Network or filename)
    ListenPort          1234                    # Network port to listen on

}


#
# This section describes how we record the framebuffer to a sequence of TGA files.
#
Video {
    # Gives the output template name (for fprintf). The first "%d" is the screen number, the second "%d" is the frame number.
    Output        "frames.tga\frame%d-%.4d.tga"

    # Background to render on
    Background    "d:\wb.tga"

    # Indicates the targetted framerate of the resulting sequence.
    Fps           25

    # If set to true, recording begins immediately. Press F9 to start recording, F10 to stop.
    Record        false

    # Sets the number of the first frame to write to.
    FirstFrame    0

    # Indicates if we want to store the background (if any) of while recording the framebuffer.
    RecordBG      no
}
```

FIG. 5.12 – Extrait d'un fichier de configuration de la libCube

Conclusion et perspectives

Bilan

L'objectif de cette thèse était de travailler sur l'interaction 3D dans le cadre de la déformation géométrique de forme libre. Ceci fait suite à un stage de DEA au cours duquel j'ai apporté quelques améliorations au modèle DOGME, sur lesquelles reposent mes travaux. Ces améliorations permettent d'utiliser des contraintes suivant un chemin quelconque, ou dotées d'un volume d'influence quelconque grâce à l'utilisation de voxels.

Afin de disposer d'un cadre applicatif concret, nous avons choisi de nous intéresser aux problèmes de modélisation géométrique dans un environnement virtuel. Grâce à cet environnement, l'utilisateur bénéficie d'une manipulation plus naturelle avec les données, lui permettant un gain de temps et et d'aisance. Nous avons pu constater que la réalité virtuelle était particulièrement adaptée à la réalisation rapide d'esquisses, notamment en dessinant «à main levée». Cependant, les environnements actuels de type «plan de travail virtuel» sont encore imposants et onéreux. Des environnements plus petits, destinés à prendre place sur un bureau, permettraient probablement une intégration massive de ces dispositifs pour les tâches de modélisation géométrique 3D. Une solution qui apparaît comme prometteuse est le *Portable Virtual Reality System* [SHPH04].

Nos travaux ont abouti à la réalisation d'un prototype de modeleur géométrique à base de contraintes, Dogme^{RV}. Ce modeleur permet de créer, d'ajuster et de travailler avec des contraintes et leurs propriétés, ainsi que des objets géométriques servant de marqueurs passifs à la visualisation de la déformation. Ce programme nous a permis d'introduire un certain nombre d'interactions sur des données abstraites, pour lesquelles la réalité virtuelle s'avère particulièrement efficace. Nous avons notamment introduit la saisie de voxels, une tâche totalement tridimensionnelle pour laquelle la perception du volume procurée par la vision stéréoscopique est cruciale. Grâce à quelques aides visuelles, le dessin en volume se fait aisément. Bien entendu la précision reste un problème à résoudre, surtout si l'on souhaite dessiner des détails très fins. Parmi les autres opérations, on peut retenir la saisie d'esquisse de courbe, qui se fait en un seul mouvement de main. L'utilisateur peut ainsi donner corps à ses idées très rapidement, ce qui optimise le processus cognitif entre l'idée et la réalisation.

La taille importante du programme et son grand nombre d'options soulèvent

également un problème de taille : contrôler une telle application requiert des techniques adaptées, plus évoluées qu'une simple fenêtre et quelques boutons d'action. Des solutions existent dans certains cas particuliers, mais peu de métaphores s'intéressent à la fois aux utilisateurs inexpérimentés et expérimentés. C'est pourquoi nous avons cherché des solutions spécifiques à ce problème, et abouti au Spin Menu.

Au cours de l'élaboration de cette métaphore, de nombreux choix on été faits et de nombreuses améliorations on été apportées afin d'optimiser l'efficacité de la technique. Nous avons d'une part apporté une solution efficace de diminution des erreurs de sélection grâce à la technique de filtrage à double intervalle. Cette technique, que l'on pourrait appliquer sans difficulté à d'autres métaphores de menu introduit entre chaque élément une zone neutre, ayant la particularité de ne pas changer l'élément sélectionné mais permettant de le valider. On évite ainsi le phénomène de «glissement» qui peut arriver lors du relâchement d'un bouton et valider un élément erroné. Les expériences réalisées avec cette technique montrent une diminution très sensible du nombre d'erreurs de sélection. D'autre part, nous avons étudié le problème de la hiérarchie, et plus particulièrement de sa représentation. Différentes options ont été mises en places et testées auprès d'utilisateurs, ce qui nous a permis de mettre en évidence plusieurs facteurs entrant en jeu dans l'efficacité de la représentation. On peut notamment retenir l'importance de la cohérence visuelle des représentations graphiques des différents niveaux, l'importance de la cohérence entre la manipulation et la représentation et la présence d'une trace claire et précise de la position courante dans la hiérarchie. Ces divers facteurs nous ont conduits à une représentation empilée des différents menus et sous-menus, qui correspond bien à l'environnement et aux critères cités.

Au final, cette métaphore offre une manipulation et une représentation simple des différentes options. Ainsi, l'application peut être aisément contrôlée par des utilisateurs expérimentés ou novices. Les nombreux tests que nous avons effectués nous ont permis d'adapter au mieux la métaphore afin d'en obtenir les meilleures performances.

Limitations et perspectives

Bien entendu, le travail est loin de s'arrêter là, aussi bien sur le Spin Menu, sur Dogme^{RV} que sur la réalité virtuelle en général.

La métaphore de menu pose spécifiquement le problème de son emplacement d'ouverture, comme la majorité des métaphores contextuelles. En effet, dans un environnement dense en objets, le menu risque d'être occulté par eux. L'utilisateur doit dans ce cas le refermer et l'ouvrir ailleurs. La transparence pourrait être utilisée pour résoudre en partie ce problème : le ou les objets qui cachent le menu peuvent être rendu translucides –ceci étant détecté à la volée. L'utilisateur continue ainsi de voir la scène, avec un minimum de changement, tout en voyant le menu.

Un problème d'ordre plus général en réalité virtuelle est le manque relatif de

précision. Lors d'une manipulation, l'utilisateur effectue tout le travail à main levée, sans pouvoir reposer sa main sur un support stable, ce qui peut nuire considérablement à la précision globale du travail. De plus, les objets de la scène virtuelle sont sans consistance solide : poser un objet sur un autre requiert une manipulation parfaitement précise, puisque aucune aide n'est fournie par l'environnement. Il pourrait être bénéfique d'adjoindre à la scène et sa manipulation un minimum de simulation physique, notamment pour les collisions. L'utilisateur pourrait alors, par exemple, placer plus aisément un point de contrainte réellement «sur» la surface de l'objet. L'utilisation de la notion de *d'accroche*[3] pourrait également s'avérer très utile : les objets attirent vers eux l'objet manipulé en deçà d'une certaine distance. On pourrait également envisager des systèmes d'accroches dynamiques, s'adaptant à la volée aux besoins de l'utilisateur.

Cependant, la précision reste problématique lorsque l'on travaille dans l'absolu, comme pour le dessin dans l'espace (pour les voxels, ou la saisie de courbe dans Dogme^{RV}). La précision ainsi que la perception de la profondeur sont ici utilisées majoritairement. On pourrait par exemple, grâce à une manipulation bi manuelle, tenir une surface de dessin dans la main non dominante –un *prop*, comme une plaque de plexiglas– sur laquelle la main dominante pourrait dessiner, comme un dessinateur maintient sa feuille de croquis. Le dessin pourrait ainsi être fait n'importe ou dans l'espace, tout en bénéficiant d'un support.

[3] dénommé snap en anglais

Bibliographie

[AB97] Fabrice Aubert and Dominique Bechmann. Animation by deformation of space-time objects. In *Eurographics*, 1997.

[ABF+04] Jérémie Allard, Edmond Boyer, Jean-Sébastien Franco, Clément Ménier, and Bruno Raffin. Marker-less real time 3d modeling for virtual reality. In *Symposium on Immersive Projection Technology*, 2004.

[AGL+03] Jérémie Allard, Valérie Gouranton, G. Lamarque, Emmanuel Melin, and Bruno Raffin. Softgenlock : Active stereo and genlock for pc cluster. In *Proceedings of the Joint IPT/EGVE'03 Workshop*, Zurich, Switzerland, May 2003.

[Bar84] Alan Barr. Global and local deformations of solid primitives. *Computer Graphics*, 18(3) :21–30, 1984.

[BB91] P. Borrel and D. Bechmann. Deformation of n-dimensional objects. *International journal of computational geometry and applications*, 1(4) :427–453, 1991.

[BBB98] Dominique Bechmann, Yves Bertrand, and Sylvain Brandel. Spécification algébrique et implantation de déformation de l'espace. *Revue Internationale de CFAO*, 13(1), 1998.

[BBT96] Dominique Bechmann, Yves Bertrand, and Sylvain Thery. Continuous free-form deformation. In *Compugraphics'96*, 1996.

[BD93] Dominique Bechmann and Nicolas Dubreuil. Animation through space and time based on a space deformation model. *The Journal of Visualization and Computer Animation*, 4(3) :165–184, 1993.

[BDHO92] Jeff Butterworth, Andrew Davidson, Stephen Hench, and Marc. T. Olano. 3dm : a three dimensional modeler using a head-mounted display. In *ACM Symposium on Interactive 3D Graphics*, pages 135–138, 1992.

[Bec95] Dominique Bechmann. *Modèle de déformation pour la modélisation géométrique et l'animation*. Habilitation à diriger des recherches, Université Louis Pasteur, December 1995.

[BG03] Dominique Bechmann and Dominique Gerber. Arbitrary shaped deformations with dogme. *The Visual Computer*, 19(2–3) :175–186, 2003.

[BH97] Doug Bowman and Larry F. Hodges. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *Symposium on Interactive 3D Graphics*, 1997.

[BJMP01] Doug Bowman, Joseph LaViola Jr., Mark Mine, and Ivan Poupyrev. Advanced topics in 3d user interface design. In *Course Notes - SIGGRAPH 2001*, 2001.

[BO71] Thomas L. Bouillon and Patrick L. Odell. *Generalized Inverse Matrix*. Wiley Inter-Science, 1971.

[Bol80] Richard Bolt. Put that here : voice and gesture at the graphics interface. In *Proceedings of ACM Siggraph'80*, pages 262–270, 1980.

[Bow99] Doug A. Bowman. *Interaction Techniques For Common Tasks In Immersive Virtual Environments - Design, Evaluation, and Application*. PhD thesis, Georgia Institute of Technology, 1999.

[BR94] Paul Borrel and Ari Rappoport. Simple constrained deformation for gemetric modeling and interactive design. *ACM Transactions on Graphics*, 13(2) :137–155, 1994.

[BW01] Doug A. Bowman and C. A. Wingrave. Design and evaluation of menu systems for immersive virtual environments. In *IEEE Virtual Reality*, 2001.

[CNSD93] C. Cruz-Neira, D. Sandin, and T. Defanti. Surround-screen projection-based virtual reality : The design and implementation of the cave. In *Proceedings of ACM SIGGRAPH'93*, pages 135–142, 1993.

[Coq90] Sabine Coquillart. Extended free-form deformations : A sculpturing tool for 3d geometric modeling. *Computer Graphics*, 24(4) :187–196, 1990.

[CSH+92a] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-dimensional widgets. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics, Special Issue of Computer Graphics, Vol. 26*, 1992.

[CSH+92b] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-dimensional widgets. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics, Special Issue of Computer Graphics, Vol. 26*, pages 183–188, 1992.

[Dar94] Rudy Darken. Hands-off interaction with menus in virtual spaces. In *Proceedings of SPIE 1994, Stereoscopic Displays and Virtual Reality Systems.*, volume 2177, pages 365–371, 1994.

[Dee95] Michael F. Deering. Holosketch : a virtual reality sketching/animation tool. In *ACM Transactions on Computer-Human Interaction (TOCHI)*, pages 220–238, 1995.

[Dor04] Geoffroy A. Dorn. Virtual reality in the energy industry (invited paper). In *Immersive Projection Technology*, 2004.

[Elk02] Mehdi Elkouhen. *Modélisation géométrique 4D appliquée à l'animation*. PhD thesis, Universite Louis Pasteur, December 2002.

[ENSK98] T. Todd Elvins, David R. Nadeau, Rina Schul, and David Kirsh. Worldlets : 3d thumbnails for 3d browsing. In *SIGCHI conference on Human factors in computing systems*, pages 163 – 170, 1998.

[FA97] Dominique Bechmann Fabrice Aubert. Volume-preserving space deformation. *Computer & Graphics*, 21(5) :625–639, 1997.

[FHZ96] A. Forsberg, K. Herdon, and R. Zeleznik. Aperture based selection for immersive virtual environments. In *ACM Symposium on User Interface software and Technology*, 1996.

[GC01] Jérome Grosjean and Sabine Coquillart. Command & control cube : a shortcut paradigm for virtual environments. In *Eurographics Workshop on Virtual Environments*, 2001.

[GD99] J. E. Gain and N. A. Dodgson. Adaptive refinement and decimation under free-form deformation. In *Proceedings of Eurographics*, 1999.

[Ger01] Dominique Gerber. Extensions au modèle de déformations dogme. Technical report, Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection, 2001.

[Gla90] Andrew Glassner, editor. *Graphic Gems I*. ACM Academic Press, 1990.

[GS99] Michael Gösele and Wolfgang Stuerzlinger. Semantic constraints for scene manipulation, 1999.

[Han97] Chris Hand. A survey of 3d interaction techniques. *Computer Graphics Forum*, 16(5) :269–281, December 1997.

[HHK92] William M. Hsu, John F. Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. *Computer Graphics*, 26(2) :177–184, 1992.

[Hop04] Don Hopkins. Pie menu central. http ://www.piemenu.com, 2004. Site web pour la démocratisation des menus circulaires.

[Huc93] L. R. Huckstep. *A Simple Guide to Orthopaedics*. Churchill Livingstone, 1993.

[JE92] R. Jacoby and S. Ellis. Using virtual menus in a virtual environment. In *SPIE : Visual Data Interpretation*, 1992.

[JF96] Q. Peng J. Feng, L. Ma. A new free-form deformation through the control of parametric surfaces. *Computer and Graphics*, 3(3) :201–214, 1996.

[KB94] G. Kurtenbach and W. Buxto. User learning and performance with marking menus. In *Proceedings of CHI '94*, pages 258–264, 1994.

[KF94] W. Krueger and B. Froehlich. The responsive workbench. *IEEE Computer Graphics and Applications*, May 1994.

[KHK94] J. Goble K. Hinckley, R. Pausch and N. Kassel. Passive real-world interface props for neurosurgical visualization. In *ACM Computer Human Interfaces*, pages 452–458, 1994.

[KHMK99] Yoshifumi Kitamura, Tomohiko Higashi, Toshihiro Masaki, and Fumio Kishino. Virtual chopsticks : Object manipulation using multiple exact interactions. In *IEEE Virtual Reality*, pages 198–204, 1999.

[KP01] Michal Koutek and Frits H. Post. Spring-based manipulation tools for virtual environments. In *Immersive Projection Technology*, 2001.

[KTY96] Kiyoshi Kiyokawa, Haruo Takemura, and Naokazu Yokoya. An empirical study on two-handed/collaborative virtual assembly. In *International Display Workshops*, 1996.

[KvHPB02] Michal Koutek, Jeroen van Hees, Frits H. Post, and A.F. Bakker. Virtual spring manipulators for particle steering in molecular dynamics on the responsiveworkbench. In *Eurographics Workshop on Virtual Environments*, pages 53–62, 2002.

[LaV00] Joseph J. LaViola. A discussion of cybersickness in virtual environments. *ACM SIGCHI Bulletin*, 32(1) :47–56, 2000.

[LCJ93] F. Lazarus, Sabine Coquillart, and P. Jancène. Interactive axial deformations. Technical Report 1891, INRIA, 1993.

[LDP+02] James Jen-Weei Lin, Henry B. L. Duh, Donald E. Parker, Habib Abi-Rached, and Thomas A. Furness. Effects of field of view on presence, enjoyment, memory and simulator sickness in a virtual environment. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 164–172, 2002.

[LG94] J. Liang and M. Green. Jdcad : A highly interactive 3d modeling system. *Computer and Graphics*, 18(4) :499–506, 1994.

[LW94] Henry J. Lamousin and Warren N. Waggenspack. Nurbs-based free-form deformations. *IEEE Computer Graphics and Applications*, pages 59–65, November 1994.

[Mac96] Ron MacCracken. Free-form deformations with lattices of arbitrary topology. In *Computer Graphics Proceedings, Annual Conference Series*, pages 181–188, 1996.

[Min95a] M. Mine. Isaac : A virtual environment tool for the interactive construction of virtual worlds. Technical Report TR95-020, University of North Carolina Computer Science, 1995.

[Min95b] M. R. Mine. Virtual environment interaction techniques. Technical Report TR95-018, Chapel Hill Computer Science, 1995.

[Min96] M. R. Mine. Working in a virtual world : Interaction techniques used in the chapel hill immersive modeling program. Technical Report TR96-029, Chapel Hill Computer Science, 1996.

[MJBF02] Tim Milliron, Robert J. Jensen, Ronen Barzel, and Adam Finkelstein. A framework for geometric warps and deformations. *ACM Transactions on Graphics*, 21(1) :20–51, January 2002.

[MJS97] Mark R. Mine, Frederick P. Brooks Jr., and Carlo H. Sequin. Moving objects in space : Exploiting proprioception in virtual-environment interaction. In *ACM Siggraph*, page A VERIFIER, 1997.

[MK99] Daniel Thalmann Marcelo Kallmann. Direct 3d interaction with smart objects. In *ACM Virtual Reality Softaware and Technology*, pages 124–130, 1999.

[MRWJ03] Michael Meehan, Sharif Razzaque, Mary C. Whitton, and Frederick P. Brooks Jr. Effect of latency on presence in stressful virtual environments. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, 2003.

[MZ99] Timothy Miller and Robert C. Zeleznik. The design of 3d haptic widgets. In *Symposium on Interactive 3D Graphics*, pages 97–102, 1999.

[OF03] Alex Olwal and Steven Feiner. The flexible pointer : a interaction techn ique for selection in augmented and virtual reality. In *IEEE Virtual Reality Annual International Symposium*, 2003.

[PB02] Hubert Peyré and Dominique Bechmann. Deformations expressed as displacement maps : An easy way to combine deformations. In *International Conference on Computational Science : workshop on Computer Graphics and Geometric Modeling*, pages 21–24, 2002.

[PBWI96] I. Poupyrev, M. Billinghurst, S. Weghorst, and T. Ichikawa. The go-go interactive technique : Non-linear mapping for direct manipulation in vr. In *ACM Symposium on User Interface Software and Technology*, 1996.

[PFC+97] J. Pierce, A. Forsberg, M. Conway, S. Hong, R. Zeleznik, and M. Mine. Image plane interaction techniques in 3d immersive environments. In *Symposium on Interactive 3D Graphics*, pages 39–44, 1997.

[PSP99] Jeffrey S. Pierce, Brian Stearns, and Randy Pausch. Two handed manipulation of voodoo dolls in virtual environments. In *Symposium on Interactive 3D Graphics*, pages 141–145, 1999.

[RH92] Warren Robinett and Richard Holloway. Implementation of flying, scaling and grabbing in virtual worlds. In *ACM Symposium on Interactive 3D Graphics*, pages 189–192, 1992.

[RN99] Romain Raffin and Marc Neveu. Déformations sous contraintes : modèle scodef étendu. In *Journées du groupe de travail Modélisation Géométrique*, september 1999.

[Rom00] Raffin Romain. *Déformations sous contraintes Généralisées*. PhD thesis, Université de Bourgogne, janvier 2000.

[RSS+02] Sharif Razzaque, David Swapp, Mel Slater, Mary C. Whitton, and Anthony Steed. Redirected walking in place. In *Eurographics Workshop on Virtual environments*, pages 123–130, 2002.

[Sch83] Ben Schneiderman. *Designing the User Interface : Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 1983.

[Sch03] Jérôme Schmid. Rapport de fin d'école d'ingénieur. Technical report, Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection, 2003.

[SCP95] R. Stoackley, M. Conway, and R. Paush. Virtual reality on a wim : Interactive worlds in miniature. In *Proceedings of CHI*, 1995.

[SES99] Dieter Schmalstieg, L. Miguel Encarnaçao, and Zsolt Szalavári. Using transparent props for interaction with the virtual table. In *SIGGRAPH Symposium on Interactive 3D Graphics*, pages 147–154, 1999.

[SF98] K. Singh and E. Fiume. Wires : a geometric deformation technique. *Computer graphics*, pages 405–414, 1998.

[SHPH04] Oliver Stefani, Hilko Hoffmann, Harshada Patel, and Frank Haselberger. Extending the desktop workplace by a portable virtual reality system. In 8^{th} *symposium on Immersive Projection Technology*, 2004.

[SIKH82] D. C. Smith, C. Irby, R. Kimball, and E. Harslem. The star user interface : an overview. In *Proceedings of AFIPS 1982 National conference*, pages 515–528, 1982.

[SN92] D. Song and M. Norman. Non-linear interactive motion control techniques for virtual space navigation. In *IEEE Virtual Reality Annual International Symposium*, pages 111–117, 1992.

[SP86] T. Sederberg and S. Parry. Free-form deformations of solid geometric models. *ACM Transactions on Graphics*, 20(4) :151–160, 1986.

[SP04] Anthony Steed and Chris Parker. 3d selection strategies for head tracked and non-head tracked operation of spatially immersive displays. In *Immersive Projection Technology*, 2004.

[SSS99] G. Smith, T. Salzman, and W. Stuerzlinger. 3d scene manipulation with constraints. In *Virtual Concept*, 1999.

[Ste03] Ludovic Sternberger. *Étude des métaphores d'interaction 3D sur le Workbench*. Mémoire de dea, Université Louis Pasteur, 2003.

[SZP89] D. J. Sturman, D. Zeltzer, and S. Pieper. Hands-on interaction with virtual environments. In *ACM SIGGRAPH symposium on User interface software and technology*, pages 19–24, 1989.

[UAW+99] Martin Usoh, Kevin Arthur, Mary C. Whitton, Rui Bastos, Anthony Steed, Mel Slater, and Jr. Frederick P. Brooks. Walking > walking-in-place > flying, in virtual environments. In *ACM Siggraph*, pages 359–364, 1999.

[WD00] G. Weshe and M. Droske. Conceptual free-form styling on the responsive workbench. In *Virtual Reality Software and Technology*, pages 83–91, 2000.

[WDC99] Kent Watsen, Rudolph Darken, and Michael Capps. A handheld computer as an interaction device to a virtual environment. In *Proceedings of the International Projection Technologies Workshop*, 1999.

[WG95] M. Wloka and E. Greenfield. The virtual tricorder : A unified interface for virtual reality. In *ACM Symposium on User Interface Software and Technology*, pages 39–40, 1995.

[WR99] Colin Ware and Jeff Rose. Rotating objects with real handles. In *ACM Computer-Human Interaction*, 1999.

[wwwa] A.r.t. gmbh. http ://www.ar-tracking.com.

[wwwb] Ascension technology corporation. http ://www.ascension-tech.com.

[wwwc] Fakespace systems inc. http ://www.fakespace.com.

[wwwd] Fifth dimension technologies. http ://www.5dt.com.

[wwwe] Immersion corp. http ://www.immersion.com.

[wwwf] Intersense inc. http ://www.isense.com.

[wwwg] Polhemus. http ://www.polhemus.com.

[YAR04] L. Yan, R.S. Allison, and S. Rushton. New simple virtual walking method - walking on the spot. In *Symposium on Immersive Projection Technology*, 2004.

[ZBM94] Shumin Zhai, William Buxton, and Paul Milgram. The silk cursor : investigating transparency for 3d target acquisition. In *SIGCHI conference on Human factors in computing systems*, pages 459–464, 1994.

[ZJFF02] Robert C. Zeleznik, Jospeh J. LaViola Jr., Daniel Acevedo Feliz, and Daniel F.Keefe. Pop through button for ve navigation and interaction. In *IEEE Virtual Reality Conference*, pages 127–134, 2002.

Troisième partie

Annexes

Annexe A

Projection perspective désaxée

Même si nos ordinateurs de bureau proposent un affichage et une interaction bidimensionnelle, beaucoup de problématiques traitées sont tridimensionnelles (conception assistée par ordinateur, modélisation, jeux vidéo, etc.). Afin de pouvoir donner une représentation des données à l'utilisateurs, il faut les projeter à l'écran, afin de les ramener à un espace 2D. Deux grandes familles de projections existent : la projection orthogonale, ou perspective.

La **projection orthogonale** est très utilisée par les architectes. Elle consiste à afficher la scène comme si elle était vue à partir d'un point situé à une distance infinie de celle-ci, avec une ouverture d'objectif infiniment petite. La distance n'intervient alors pas dans le processus de projection, et les lignes parallèles dans l'espace 3D restent parallèles sur la projection.

La **projection perspective** est très utilisée par l'animation et le jeu vidéo. Elle consiste à simuler une prise de vue de la scène par un objectif dont la position est connue. La distance intervient dans le processus de projection, si bien que deux lignes parallèles dans l'espace 3D ne sont pas forcément parallèles sur la projection. Ces lignes non parallèles se recoupent en un ou plusieurs points, que l'on nomme *points de fuite*.

Nous nous concentrerons ici sur la projection perspective plutôt que la projection orthogonale, car donne une meilleure approximation de notre perception de la réalité. Cependant, dans le cas de la projection sur les écrans d'un environnement virtuel, la projection perspective prend une forme inhabituelle, que nous allons détailler maintenant.

Le plus souvent, la projection perspective consiste à réduire l'intérieur d'un tronc de pyramide rectangle en un plan (l'écran). Ce tronc de pyramide représente la portion d'espace qui sera «visible» de la scène. L'expression d'une telle matrice est rappelée en A.1, avec les variables illustrées Figure A.1. Cette approche est majoritairement utilisée par le matériel graphique, notamment OpenGL, pour accélérer le rendu.

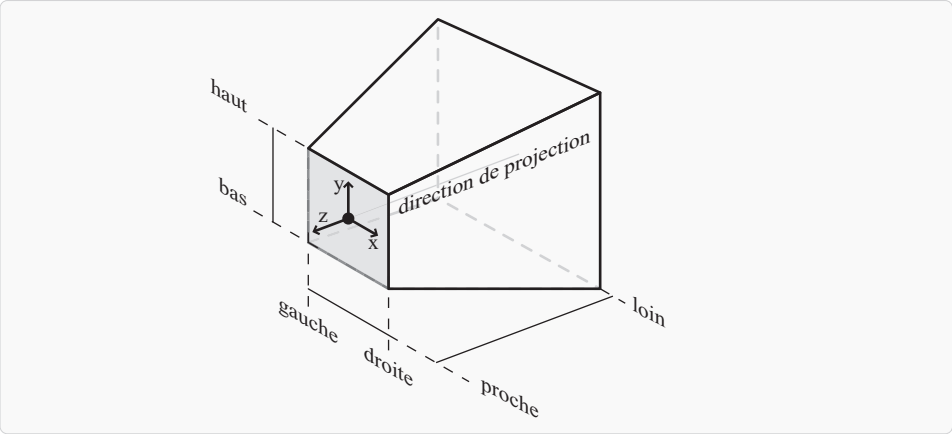


FIG. A.1 – Le cône de projection perspective.

$$M_h = \begin{pmatrix} \frac{2*proche}{droite-gauche} & 0 & \frac{droite+gauche}{droite-gauche} & 0 \\ 0 & \frac{2*proche}{haut-bas} & \frac{haut+bas}{haut-bas} & 0 \\ 0 & 0 & \frac{loin+proche}{loin-proche} & \frac{2*loin*proche}{loin-proche} \\ 0 & 0 & -1 & 1 \end{pmatrix} \quad \text{(A.1)}$$

Dans ce cas de figure, les grandeurs $proche, loin, haut, bas, droite$ et *gauche* sont choisies arbitrairement par le programmeur. Au final, il se produit toujours que *haut* corresponde au haut de l'écran, $droite$ corresponde à la droite, et similairement pour *bas* et *gauche*. Ce que l'on cherche à faire ici est d'aplatir le contenu de la pyramide sur le plan *proche*. Cette formulation cache l'hypothèse implicite que la projection se fait «le long» de l'axe des Z, c'est à dire que l'objectif regarde dans cette direction. Du coup, l'utilisateur doit lui aussi regarder dans la même direction –c'est à dire, regarder son écran bien en face– afin d'avoir la meilleure sensation possible.

En réalité virtuelle, lorsque des écrans entourent l'utilisateur, cette hypothèse n'est pas forcément vraie : l'utilisateur peut regarder un écran à 45°, l'autre à 70°, etc. Pourtant, l'ensemble des écrans doit offrir une représentation cohérente de la scène virtuelle. On souhaite bien entendu un affichage perspective, plus proche de notre perception quotidienne de la réalité. Il s'agit donc de pouvoir exprimer la pyramide de projection par rapport à un écran situé dans un plan arbitraire et une position arbitraires de l'espace.

Afin de bénéficier de l'accélération matérielle de la projection, nous devons nous ramener cette situation à la situation «classique» du plan de projection porté par le plan XY. Ceci peut se faire de manière simple par une matrice de passage, permettant de ramener le plan de l'écran sur le plan XY. Notons qu'en procédant

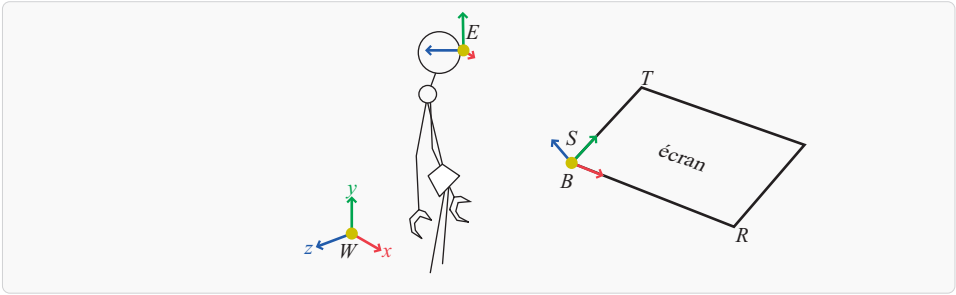


FIG. A.2 – Configuration de l'espace.

ainsi, la pyramide est également transformée, et n'est en général pas centrée sur l'axe de projection. Heureusement, cela n'est pas un problème.

Définissons tout d'abord quelques notations.

Base Une base est définie par un trièdre orthonormé ainsi qu'un point origine. La base globale, dont les axes sont alignés avec les axes globaux et situés en O est notée W. Les différents axes d'une base B base sont référencés par B_x, B_y et B_z.

Point Notons P_A les coordonnées du point P exprimé dans la base A.

Coordonnées Notons P_{A_x} la coordonnée x du point P_A.

Matrice de passage Notons $M_{a\to b}$ la matrice permettant d'exprimer les coordonnées d'un point P_A dans la base B. Ainsi, $P_B = M \cdot P_A$, et inversement $P_A = M^{-1} \cdot P_B$.

Imaginons que nous ayons un utilisateur dans un environnement avec suivi de mouvement. Soit E la base positionnée sur l'oeil de l'utilisateur, et orienté selon sa direction de regard. Soit S la base positionnée sur le coin inférieur gauche de l'écran, et orientée de sorte que l'écran soit porté par le plan XY dans S. Nommons T, B et R le point haut gauche, bas gauche et bas droit de l'écran, respectivement. La figure A illustre ces points.

Afin de pouvoir appliquer une projection perspective habituelle projetant la scène sur le plan XY de B, il faut transformer l'ensemble de la scène au préalable afin de faire correspondre le plan de projection avec le plan XY de W. Afin de conserver la perspective correcte, la pyramide doit partir de l'oeil de l'utilisateur, qui doit donc subir la même transformation. Ceci s'exprime parfaitement par la matrice de passage $P_{S\to W}$, que l'on obtient comme énoncé dans l'équation A.2.

$$P_{S\to W} = \begin{bmatrix} S_{x_x} & S_{y_x} & S_{z_x} & E_x \\ S_{x_y} & S_{y_y} & S_{z_y} & E_y \\ S_{x_z} & S_{y_z} & S_{z_z} & E_z \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \tag{A.2}$$

Ainsi, la matrice de projection sur l'écran considéré sera le produit de la matrice $P_{S\to W}$ par une matrice de projection M_h habituelle. Il reste cependant un petit détail avant d'être en mesure d'effectuer la projection : par définition, la projection «écrase» une pyramide sur un plan (l'écran d'arrivée). Or, nous avons considéré jusqu'ici que nous projetions sur l'oeil, qui est un point. Ceci est impossible. Nous devons pour cela placer un écran «virtuel», proche de l'oeil de l'utilisateur, sur lequel projeter, afin d'être en mesure de fournir les coordonnées $gauche, droite, haut$ et bas de la construction de la pyramide de projection. Ces coordonnées sont obtenues en projetant «manuellement» les quatre coins de l'écran (exprimé dans la base W) sur un plan situé à $proche$ unités de l'origine selon W_z, comme explicité dans l'équation A.3.

$$\begin{cases} droite = \frac{(R_S * P_{S\to W})_x * proche}{(B_S * P_{S\to W})_z} \\ gauche = \frac{(B_S * P_{S\to W})_x * proche}{(B_S * P_{S\to W})_z} \\ haut = \frac{(T_S * P_{S\to W})_y * proche}{(B_S * P_{S\to W})_z} \\ bas = \frac{(B_S * P_{S\to W})_y * proche}{(B_S * P_{S\to W})_z} \end{cases} \tag{A.3}$$

Nous disposons ainsi de tous les éléments nécessaires à la projection sur l'écran considéré en utilisant une projection perspective standard :

$$projection = M_h(gauche, droite, haut, bas, proche, loin) * P_{S\to W}$$

Cette matrice est fournie à OpenGL dans sa matrice `GL_PROJECTION`, au lieu de la matrice M_h habituelle. La figure A.3 donne le code permettant de calculer cette matrice issu de la libCube.

Notons que cette formulation a tout de même un inconvénient majeur, ainsi qu'un inconvénient mineur. L'utilisation d'un plan de projection «virtuel» pose problème si celui-ci se trouve derrière le plan de l'écran, c'est à dire si l'utilisateur s'approche trop près de l'écran. Il faut également veiller à utiliser une distance *proche* suffisamment faible pour que cette situation ne se produise qu'exceptionnellement ou jamais. L'autre inconvénient est que cette formulation ne laisse apparaître nulle part de notion d'ouverture de la pyramide de projection, comme c'est souvent le cas dans la projection sur écran 2D. Ici, les écrans servent réellement de «fenêtres» par lesquelles on verrait la scène virtuelle à sa taille réelle, contrairement au classique paradigme d'un objectif «filmant» la scène.

```
// Construction de la matrice de projection désaxée pour
// un écran dont les coordonnées des coins en bas à droite,
// gauche et en haut à gauche sont connus dans le repère
// global.

void setupProjection (
            const CVertex &screenBottomLeft,
            const CVertex &screenBottomRight
            const CVertex &screenTopLeft,
            const CVertex &headPos,
            const float    eyeSeparation,
            const float    zClipNear,
            const float    zClipFar
            )
{

  CVertex   screenLeft   =
            (screenBottomRight-screenBottomLeft).Normalize (),
            screenUp     =
            (screenBottomLeft-screenTopLeft).Normalize (),
            screenFront  = (screenLeft ^ screenUp);
  CMatrix  mRepere,
            mRepereInv;

  // Remplissage de la matrice par colonnes
  mRepere.SetCol (0, screenLeft,                    0);
  mRepere.SetCol (1, screenUp,                      0);
  mRepere.SetCol (2, screenFront,                   0);
  mRepere.SetCol (3, headPos + screenLeft*eyeOffset, 1);
  mRepereInv = mRepere.Invert ();

  // Calcul des coordonnées de l'écran en coordonnées
  // utilisateur
  CVertex   bottomLeft  = mRepereInv * screenBottomLeft;
  CVertex   bottomRight = mRepereInv * screenBottomRight;
  CVertex   topLeft     = mRepereInv * screenTopLeft;

  // Maintenant, remplissage de la matrice de projection
  // d'OpenGL
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();

  glFrustum (
    bottomLeft.x  * zClipNear / fabsf (bottomLeft.z),
    bottomRight.x * zClipNear / fabsf (bottomLeft.z),
    bottomLeft.y  * zClipNear / fabsf (bottomLeft.z),
    topLeft.y     * zClipNear / fabsf (bottomLeft.z),
    zClipNear,
    zClipFar
   );

  glMultMatrixf (mRepereInv);
}
```

FIG. A.3 – Code de calcul de la projection désaxée

Annexe B

Données expérimentales

B.1 Test des fonctions de filtrage

B.1.1 Temps de sélection, «aussi vite que possible» (Tab. B.1)

Ce tableau présente les temps de sélection effectués par les utilisateurs sous la condition d'aller aussi vite que possible.

B.1.2 Temps de sélection, «aussi précis que possible» (Tab. B.2)

Ce tableau présente les temps de sélection effectués par les utilisateurs sous la condition d'être aussi précis que possible.

B.2 Test du nombre d'éléments du spin menu

B.2.1 Temps de sélection (Tab. B.3)

Ce tableau présente les temps de sélection effectués par les utilisateurs pour la détermination du nombre maximal d'éléments. Aucune fonction de filtrage n'avait été appliquée. Deux tests avaient été faits pour chaque nombre d'élément : en fixant la taille angulaire d'un élément à 8 degrés (*Adapté*), ou en fixant la taille de l'anneau à 160 degrés et en répartissant les éléments dessus (*Fixe*).

B.3 Test de comparaison du Spin, C^3 et Fenêtres

B.3.1 Temps de sélection (Tab. B.4)

Ce tableau comporte les temps moyens de tous les utilisateurs sur chaque métaphore testée, arrondis au centième de seconde.

		Sans		Magnetique		Double Intervalle		Double Intervalle Ex		RingBar	
#	Item	Temps	Err (%)	Temps	Err (%)	Temps	Err (%)	Temps	Err (%)	Temps	Err (%)
0	7	2.06	9	2.14	-	2.21	-	1.91	13	1.27	-
1	2	1.54	27	1.55	-	1.31	50	1.22	13	1.23	13
2	7	1.30	45	1.19	18	2.93	-	1.07	13	0.97	-
3	6	1.75	45	1.58	9	2.66	8	1.21	-	1.12	13
4	4	1.12	18	0.98	18	1.05	-	0.84	13	0.96	-
5	3	1.27	-	1.14	18	1.06	8	1.06	-	0.96	13
6	6	1.38	27	1.23	-	1.27	17	1.14	-	0.96	13
7	1	1.30	18	1.23	-	1.36	8	1.03	-	1.02	-
8	6	1.32	9	1.29	-	1.25	8	1.21	13	0.88	-
9	7	1.35	18	1.29	-	1.19	-	1.03	25	1.00	-
10	1	1.41	18	1.27	-	1.20	25	1.19	-	1.04	-
11	6	1.42	27	1.28	9	1.38	8	1.43	13	0.98	-
12	7	1.29	-	1.38	9	1.22	8	1.10	-	1.03	13
13	1	1.22	-	1.29	-	1.16	17	1.13	-	1.06	13
14	2	1.62	18	1.47	-	1.43	17	1.27	-	1.11	-
15	7	1.28	-	1.24	-	1.19	8	1.12	13	1.04	-
16	1	1.28	-	1.26	-	1.20	-	1.14	-	0.97	25
17	5	1.24	9	1.12	18	1.17	-	0.97	-	0.93	-
18	3	1.18	-	1.11	9	1.08	8	0.93	-	0.98	-
19	4	1.09	9	0.99	27	1.10	-	0.88	-	0.96	-
20	7	1.20	18	1.18	18	1.25	8	1.06	-	1.01	-
21	6	1.51	9	1.74	9	1.31	25	1.32	-	1.05	-
22	7	1.45	18	1.27	-	1.21	8	1.09	13	1.07	-
23	6	1.76	-	1.39	9	1.30	-	1.19	13	1.17	13
24	7	1.24	18	1.24	-	1.17	8	1.09	13	1.03	-
25	2	1.56	36	1.45	27	1.39	-	1.36	-	1.10	-
26	6	1.18	27	1.28	18	1.25	17	1.33	13	1.29	-
27	2	1.34	27	1.20	27	1.24	-	1.11	-	0.98	-
28	3	1.30	9	1.43	-	1.22	-	1.10	-	1.08	-
29	1	1.30	27	1.26	9	1.21	8	1.13	-	1.02	-
30	5	1.26	18	2.18	9	1.46	17	1.02	13	0.91	-
31	6	1.35	36	1.34	27	1.32	-	1.02	-	0.87	-
32	1	1.44	27	1.40	-	1.12	8	1.07	-	0.86	25
33	5	1.20	-	1.12	27	1.47	8	0.98	-	0.85	-
34	4	1.02	-	1.04	9	0.95	8	0.93	13	0.87	-
35	5	1.15	-	1.10	-	1.03	8	0.97	-	0.82	-
36	3	1.04	-	1.23	9	1.05	-	0.87	-	0.87	-
37	6	1.35	45	1.17	27	1.31	-	1.08	-	0.86	-
38	3	1.16	-	1.15	-	1.13	8	0.93	25	0.86	-
39	1	1.35	18	1.34	-	1.14	-	1.04	-	1.07	-
40	5	1.19	-	1.19	18	1.13	-	0.96	13	0.89	-
41	2	1.32	27	1.16	18	1.33	25	1.13	-	0.93	-
42	5	1.28	9	1.18	-	1.04	-	0.87	-	0.79	-
43	2	1.38	27	1.15	27	1.22	-	1.12	-	0.95	-
44	1	1.55	18	1.32	27	1.21	17	1.04	13	1.16	-
45	5	1.32	9	1.17	9	1.25	-	1.16	13	1.01	-
46	4	1.17	-	1.02	-	1.12	8	0.85	13	0.81	-
47	1	1.26	9	1.24	-	1.27	8	0.99	-	0.99	-
48	2	1.38	18	1.39	36	1.47	25	1.27	-	1.11	-

TAB. B.1 – Temps moyens, fonctions de filtrage, aussi vite que possible

		Sans		Magnetique		Double Intervalle		Double Intervalle Ex		RingBar	
#	Item	Temps	Err (%)	Temps	Err (%)	Temps	Err (%)	Temps	Err (%)	Temps	Err (%)
0	7	2.61	-	2.07	-	2.38	-		14	2.09	-
1	2	2.32	27	1.76	27	1.76	-	1.39	-	1.32	-
2	7	1.91	-	1.59	9	1.63	-	1.45	-	1.27	-
3	6	2.06	-	1.83	18	1.78	-	1.40	-	1.32	-
4	4	1.41	9	1.12	18	1.25	8	0.94	-	1.04	-
5	3	1.45	-	1.29	-	1.45	-	1.09	-	1.20	13
6	6	1.67	-	1.46	-	1.70	8	1.24	-	1.18	-
7	1	1.75	-	1.61	-	1.64	-	1.24	14	1.35	-
8	6	1.67	18	1.71	9	1.75	-	1.38	-	1.16	13
9	7	1.85	-	1.82	-	1.55	-	1.24	-	1.48	-
10	1	1.88	9	1.77	18	1.60	8	1.30	14	1.43	-
11	6	1.91	27	1.77	-	1.64	-	1.46	-	1.30	-
12	7	1.96	9	1.76	-	1.53	-	1.27	-	1.28	-
13	1	2.00	-	1.78	-	1.48	8	1.35	-	1.36	-
14	2	2.14	18	1.75	-	1.88	-	1.47	-	1.43	-
15	7	1.76	9	1.58	9	1.64	-	1.23	-	1.23	-
16	1	2.02	18	1.69	-	1.60	8	1.38	-	1.27	-
17	5	1.51	9	1.43	-	1.43	-	1.34	-	1.05	-
18	3	1.47	-	1.28	-	1.29	-	1.08	-	1.07	-
19	4	1.26	-	1.13	-	1.01	-	1.04	-	0.99	-
20	7	1.84	-	1.60	-	1.49	-	1.31	-	1.23	-
21	6	2.05	-	1.66	9	1.77	8	1.63	-	1.26	-
22	7	1.81	-	1.74	-	1.45	8	1.29	-	1.30	-
23	6	1.86	-	1.74	-	1.64	-	1.42	-	1.32	-
24	7	1.77	9	1.54	9	1.51	-	1.23	-	1.24	13
25	2	1.83	18	1.73	-	1.73	8	1.54	-	1.29	-
26	6	1.65	-	1.52	-	1.51	-	1.32	-	1.09	-
27	2	1.68	9	1.61	-	1.57	-	1.38	-	1.18	-
28	3	1.53	-	1.40	-	1.44	-	1.31	-	1.22	-
29	1	1.72	27	1.62	-	1.65	-	1.31	29	1.36	-
30	5	1.34	36	1.31	9	1.49	-	1.21	-	1.14	-
31	6	1.88	9	1.61	-	1.63	-	1.35	-	1.16	-
32	1	1.73	-	1.66	-	1.56	-	1.30	-	1.31	-
33	5	1.49	-	1.55	-	1.38	-	1.11	-	0.98	-
34	4	1.25	-	1.09	-	1.13	-	0.88	-	0.93	-
35	5	1.31	-	1.31	-	1.35	-	1.07	-	1.00	-
36	3	1.34	-	1.21	-	1.26	-	1.03	-	1.07	-
37	6	1.71	-	1.55	-	1.41	-	1.36	-	1.16	-
38	3	1.30	18	1.32	-	1.35	17	1.13	-	1.08	-
39	1	1.93	-	1.62	-	1.56	-	1.29	-	1.29	-
40	5	1.62	-	1.45	18	1.52	-	1.12	-	1.05	-
41	2	1.57	18	1.55	-	1.69	-	1.41	-	1.22	-
42	5	1.47	-	1.26	9	1.47	-	1.01	-	1.05	-
43	2	1.61	9	1.75	-	1.58	-	1.32	-	1.26	-
44	1	2.06	18	1.74	-	1.56	-	1.31	-	1.43	-
45	5	1.51	9	1.46	-	1.45	-	1.22	-	1.07	-
46	4	1.14	-	1.28	-	1.08	-	0.93	-	1.06	-
47	1	1.57	-	1.48	18	1.38	-	1.17	-	1.28	-
48	2	1.85	-	1.71	-	1.66	-	1.42	-	1.43	-

TAB. B.2 – Temps moyens, fonctions de filtrage, aussi précis que possible

5 éléments				
	Taille adaptée		Taille fixe	
Item	Temps	Err (%)	Temps	Err (%)
2	2.14	22	1.84	56
3	1.23	-	1.23	11
5	1.49	-	2.11	22
1	1.36	-	1.91	-
5	1.50	11	2.01	11
4	1.48	-	1.61	-
3	1.11	-	1.05	-
5	1.31	11	1.43	11
1	1.47	-	1.68	11
2	1.61	-	1.64	-
3	1.29	-	1.04	-
2	1.28	-	1.35	11
1	1.63	-	1.51	22
3	1.22	11	1.32	-
2	1.24	-	1.34	-
5	1.33	11	1.54	11
3	1.18	-	1.15	-
4	1.36	22	1.32	22
3	1.26	-	1.34	-
2	1.53	-	1.36	11
4	1.44	11	1.31	11
1	1.45	11	1.61	-
3	1.23	-	1.22	11
2	1.35	-	1.34	-
4	1.42	11	1.36	11
5	1.35	11	1.50	11
3	1.23	-	1.11	11
5	1.43	11	1.62	22
1	1.64	11	1.98	-
5	1.50	-	1.60	-
4	1.62	-	1.53	11
2	1.25	11	1.44	-
3	1.26	-	1.27	-
4	1.34	11	1.34	11
5	1.35	11	1.39	22
2	1.43	-	1.44	-
4	1.28	33	1.42	11
3	1.24	-	1.17	-
5	1.51	11	1.75	11
3	1.29	-	1.14	-
5	1.52	33	1.60	11
4	1.46	11	1.59	11
2	1.27	-	1.37	-
5	1.56	11	1.58	22
4	1.43	-	1.40	11
2	1.38	-	1.48	-
1	1.37	22	1.56	22
3	1.25	-	1.18	-
4	1.38	11	1.29	11
2	1.42	-	1.38	-

7 éléments				
	Taille adaptée		Taille fixe	
Item	Temps	Err (%)	Temps	Err (%)
7	1.83	11	3.39	33
2	2.08	22	1.92	22
7	1.45	22	1.82	-
6	1.70	-	1.64	22
4	1.58	-	1.36	-
3	1.35	-	1.36	-
6	1.63	-	1.80	22
1	1.63	-	1.62	11
6	1.57	-	1.54	33
7	1.59	-	1.78	-
1	1.45	11	1.80	-
6	1.62	-	1.55	33
7	1.71	-	1.57	22
1	1.54	22	1.92	11
2	1.87	11	1.87	-
7	1.61	-	2.01	-
1	1.61	22	1.79	-
5	1.51	22	1.63	-
3	1.38	11	1.34	-
4	1.39	11	1.09	-
7	1.49	-	1.57	-
6	1.71	22	1.75	11
7	1.52	11	1.58	22
6	1.58	-	1.73	11
7	1.54	-	1.52	11
2	1.78	22	1.76	22
6	1.50	-	1.63	22
2	1.49	-	1.53	11
3	1.43	-	1.61	-
1	1.76	11	1.68	11
5	1.62	-	1.75	-
6	1.70	22	1.71	11
1	1.44	22	1.52	22
5	1.25	22	1.45	-
4	1.22	-	1.42	11
5	1.49	11	1.37	11
3	1.41	-	1.64	-
6	1.66	33	2.16	11
3	1.40	11	1.51	22
1	1.57	-	1.76	11
5	1.35	33	1.47	-
2	1.82	22	1.63	11
5	1.45	-	1.47	-
2	1.68	11	1.60	22
1	1.45	-	1.84	11
5	1.52	-	1.76	11
4	1.30	-	1.45	-
1	1.50	-	1.54	11
2	1.66	33	1.93	22
5	1.40	-	1.55	11

9 éléments				
	Taille adaptée		Taille fixe	
Item	Temps	Err (%)	Temps	Err (%)
6	2.09	-	1.95	-
9	1.60	25	1.95	13
8	1.91	25	1.90	13
5	1.16	13	1.36	13
9	1.42	13	1.33	25
2	1.98	13	1.45	38
4	1.50	-	1.68	-
1	1.87	13	1.85	38
8	1.66	38	1.94	13
3	1.61	25	1.84	13
9	1.54	-	1.86	-
3	1.49	-	2.06	25
8	1.82	25	1.66	38
7	1.62	-	1.50	25
8	1.79	13	1.79	13
6	1.61	-	1.60	13
8	1.59	25	1.57	50
9	1.97	-	2.76	13
4	1.73	-	1.70	-
1	1.76	13	2.19	-
7	1.58	25	2.42	-
6	1.51	-	1.45	-
1	1.61	13	1.52	13
5	1.22	-	1.41	13
8	1.39	25	1.62	-
7	1.66	13	1.94	13
6	1.38	-	1.51	-
9	1.52	-	1.90	-
6	1.71	13	1.82	-
3	1.37	25	1.83	13
1	1.75	13	1.75	-
3	1.63	13	1.95	13
1	1.79	-	1.79	-
7	1.55	13	1.85	-
5	1.44	-	1.41	-
9	1.51	13	1.43	25
2	2.04	38	1.78	38
8	1.88	13	2.07	25
4	1.47	-	1.44	-
3	1.51	13	1.85	38
7	1.69	-	1.49	25
3	1.52	13	1.38	25
4	1.46	13	1.42	-
7	1.67	13	1.94	25
3	1.40	-	1.49	13
4	1.35	13	1.56	-
8	1.87	-	2.05	13
3	1.41	25	1.68	13
2	1.61	25	1.73	38
6	1.56	-	1.74	25

11 éléments				
	Taille adaptée		Taille fixe	
Item	Temps	Err (%)	Temps	Err (%)
9	2.47	11	1.57	33
10	2.07	44	2.08	33
2	2.43	33	2.14	11
8	1.63	11	1.58	11
6	1.34	-	1.47	-
11	1.72	11	2.13	-
2	2.19	22	2.04	-
1	1.95	11	1.91	11
8	1.79	11	1.58	11
6	1.30	-	1.32	-
9	1.95	-	1.54	22
7	1.39	-	1.35	11
8	1.60	11	1.52	22
4	1.58	22	1.51	-
8	1.57	11	1.82	-
10	2.18	11	1.91	22
3	1.83	11	1.98	11
8	1.68	-	1.66	-
9	2.06	-	1.92	11
11	1.99	-	1.87	-
7	1.52	33	1.61	-
8	1.55	-	1.79	22
9	1.67	11	1.86	22
6	1.37	11	1.41	-
7	1.61	-	1.41	-
8	1.48	-	1.67	-
3	1.82	11	2.14	44
2	2.24	44	2.48	11
10	2.08	33	2.44	11
11	1.81	-	1.97	-
3	1.97	11	1.84	11
5	1.57	11	1.87	22
10	1.83	22	2.34	11
8	1.67	11	1.74	-
6	1.37	-	1.26	-
5	1.37	-	1.26	22
8	1.59	-	1.62	-
6	1.20	-	1.32	-
7	1.34	-	1.47	11
8	1.61	22	1.50	22
1	1.87	11	2.15	11
6	1.40	-	1.54	11
1	1.62	-	2.13	11
6	1.35	-	1.42	-
5	1.49	-	1.43	-
6	1.22	-	1.25	-
2	2.00	44	1.98	33
4	1.65	-	1.77	-
11	1.77	11	1.87	-
7	1.46	-	1.52	-

15 éléments				
	Taille adaptée		Taille fixe	
Item	Temps	Err (%)	Temps	Err (%)
12	2.08	56	3.44	56
3	2.49	-	2.38	33
5	2.06	-	2.51	11
11	1.84	11	1.88	-
15	2.02	11	1.84	-
5	2.10	11	2.09	44
4	1.97	-	2.08	-
8	1.36	11	1.40	-
15	1.69	-	1.80	-
6	1.72	11	1.75	-
2	2.77	22	2.72	22
13	2.75	-	3.27	22
2	2.68	-	2.20	22
12	2.25	-	2.21	11
11	1.93	22	1.98	-
3	1.94	11	2.12	11
13	2.29	22	1.89	33
7	1.72	11	2.36	33
10	1.78	11	1.88	-
3	2.20	-	2.08	22
4	2.28	33	2.06	-
8	3.54	-	1.42	11
7	1.36	-	1.63	11
12	2.48	44	1.76	22
9	1.63	-	1.64	11
6	1.65	-	1.74	-
3	2.61	22	2.03	22
7	1.55	-	1.47	-
12	1.85	33	1.68	11
4	2.00	-	2.21	44
10	1.56	-	1.72	11
8	1.27	-	1.23	-
3	2.50	11	1.83	11
5	2.40	-	1.82	22
11	1.89	11	2.10	22
10	1.98	-	1.75	-
9	1.44	-	1.52	11
7	1.54	-	1.62	11
3	2.29	11	2.23	11
4	2.13	-	1.97	22
15	1.92	44	2.07	-
7	1.93	-	1.54	-
2	2.43	56	2.28	-
9	1.75	22	1.76	11
14	2.03	33	2.26	33
3	2.34	11	2.11	11
15	2.25	-	1.90	-
8	1.38	-	1.28	-
13	2.11	11	1.93	11
5	1.82	-	2.01	22

TAB. B.3 – Temps moyens en fonction du nombre d'éléments

Menu 2D	Boutons 2D	Fenêtres 3D	Ccube	Anneau
1.15	0.89	1.24	2.33	1.21
1.08	0.74	0.95	1.96	1.22
0.98	0.83	1.18	1.81	1.07
0.92	0.93	1.40	1.63	1.21
0.83	0.89	1.37	1.52	0.84
0.78	0.84	1.17	1.68	1.06
0.91	0.90	1.27	1.71	1.14
0.92	0.73	1.06	1.67	1.03
1.01	0.59	1.00	1.59	1.21
0.95	0.79	1.14	1.64	1.03
0.95	0.78	1.25	1.66	1.19
0.86	0.89	1.33	1.72	1.43
0.88	0.78	1.20	1.69	1.10
0.96	0.77	1.07	1.61	1.13
0.80	0.72	1.12	1.32	1.27
0.90	0.72	1.01	1.66	1.12
0.83	0.74	1.14	1.68	1.14
0.89	0.75	1.08	1.59	0.97
0.84	0.77	1.23	1.45	0.93
0.82	0.68	1.03	1.57	0.88
0.92	0.64	1.00	1.65	1.06
0.98	0.92	1.34	1.79	1.32
1.02	0.78	1.05	1.37	1.09
0.87	0.73	1.08	1.78	1.19
0.88	0.81	1.14	1.64	1.09
0.95	0.65	1.02	1.48	1.36
0.97	0.82	1.18	1.48	1.33
0.84	0.71	1.16	1.76	1.11
0.78	0.73	1.09	1.81	1.10
0.78	0.88	1.29	1.38	1.13
0.91	0.81	1.08	1.86	1.02
0.88	0.83	1.37	1.57	1.02
0.97	0.80	1.13	1.37	1.07
0.92	0.74	1.11	1.51	0.98
0.74	0.74	1.09	1.79	0.93
0.78	0.64	1.03	1.80	0.97
0.74	0.62	0.95	1.56	0.87
0.98	0.81	1.23	1.64	1.08
0.80	0.79	1.27	1.52	0.93
0.83	0.80	1.23	1.65	1.04
0.85	0.86	1.15	1.50	0.96
0.80	0.73	1.11	1.45	1.13
0.90	0.72	1.08	1.50	0.87
0.76	0.71	1.06	1.25	1.12
0.72	0.77	1.17	1.44	1.04
0.95	0.68	1.13	1.89	1.16
0.77	0.80	1.11	1.45	0.85
0.77	0.64	0.96	1.90	0.99
0.70	0.82	1.16	1.20	1.27

TAB. B.4 – Comparaison entre métaphores

B.3.2 Questionnaire (Fig. B.1)

Cette figure représente le questionnaire auquel ont répondu les utilisateurs ayant participé au test de manipulation sur Dogme^{RV}. Les indications qui leur ont été données oralement sont données ci dessous.

Appréciation générale L'impression générale que vous a procuré la technique dans son ensemble : avez-vous aimé ou non.

Facilité de localisation des éléments Comment jugez-vous la clarté de la représentation graphique ? Peut-on bien localiser l'élément que l'on cherche ? On y voit parfaitement clair ?

Intuitivité de la manipulation Est-il facile de comprendre comment utiliser la technique ?

Aisance de la manipulation Est-il facile de piloter la technique ?

Effet mémoire Est-ce que vous avez l'impression que l'habitude peut vous aider à utiliser mieux la technique ?

B.3.3 Notation des métaphores (Fig. B.2)

Le tableau suivant donne les différentes notes attribuées par les utilisateurs ayant passé le test de comparaison d'interfaces sur Dogme^{RV}. Le questionnaire lui-même est présenté à la section B.3.2.

B.4 Test de la représentation hiérarchique

Le tableau B.3 récapitule les temps de sélection moyens, dans chaque niveau, dans un menu à 4 niveaux hiérarchique. La consigne donnée aux utilisateurs était d'aller aussi vite qu'ils se sentaient capable de le faire en essayant de ne pas se tromper.

B.5 Manipulation sur Dogme^{RV}

La figure B.4 donne le détail des opérations à effectuer par les utilisateurs lors du test de comparaison des interfaces de contrôle d'application sur Dogme^{RV}. Chaque instruction écrite était accompagnée d'une instruction similaire (contenant éventuellement des informations plus détaillées) audio préenregistrée.

Questionnaire individuel

Evaluation des métaphores d'interation *Icones, Ccube* et *Ring*

Informations générale

Nom & Prénom Age Sexe

Profession

Vous utilisez un ordinateur…
- ❑ Jamais
- ❑ Une fois par mois
- ❑ Une fois par semaine
- ❑ Tous les jours

Vous pratiquez les jeux vidéo…
- ❑ Jamais
- ❑ Une fois par mois
- ❑ Une fois par semaine
- ❑ Tous les ours

Les tests que vous venez de passer

Donnez une note entre 1 (mauvais) et 5 (bon) :

	Menu à icônes	**Menu en boîte**	**Menu en anneau**
Appréciation générale			
Facilité de localisation des éléments			
Intuitivité de la manipulation			
Aisance de la manipulation			
Effet mémoire			

Commentaires

Encore merci de votre participation !

FIG. B.1 – Questionnaire

					Fenêtres 3D					Ccube					Menu circulaire				
Age	Sexe	Utilisation	Gamer	Debutant ?	Appréciation	Localisation	Intuitivité	Aisance	Mémoire	Appréciation	Localisation	Intuitivité	Aisance	Mémoire	Appréciation	Localisation	Intuitivité	Aisance	Mémoire
30	M	4	2	N	4	5	5	3	4	3	5	4	4		4	5	5	5	5
26	M	4	2	O	4	4	5	4	5	4	3	4	4	5	2	5	5	2	
30	M	4	2	O	4	5	5	3	5	4	4	4	4		5	3	3	4	3
12	M	1	4	O	4	5	5	5	5	2	2	5	2	4	3	4	5	3	4
28	M	4	4	O	4	4	5	5	1	1	1	4	2		4	5	3	4	
22	M	4	3	N	4	4	5	5	5	2	2	3	1	2	5	3	3	5	5
24	M	4	3	O	4	4	5	5	5	4	4	4	3		4	5	4	3	5
25	M	4	4	N	5	5	5	5	5	3	3	3	3	4	4	3	3	4	4
43	M	4	3	O	5	5	5	5		3	5	2	2		3	4	4	4	4
50	M	4	1	O	4	5	5	4	4	3	3	4	4		4	5	5	4	4
24	M	4	3	N	5	4	5	5	5	4	2	4	3	4	2	2	3	3	
33	M	4	3	N	4	5	4	5	4	2	4	2	4	4	4	4	3	4	4
28	F	4	2	O	5	5	5	5	5	1	3	3	4	5	5	5	4	5	5
39	F	4	1	N	3	3	4	3	3	3	3	4	3	4	4	3	4	4	4
25	F	4	2	N	4	4	5	4	5	3	2	4	3	3	4	4	4	3	4
23	F	4	1	O	4	5	5	5	4	3	3	5	4		5	4	5	4	4
48	F	3	1	O	3	5	5	5	4	4	4	4	2	2	5	5	5		
50	F	4	1	O	3	5	5	4		3	2	5	3		5	4	5	3	4
36	F	4	1	O	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

FIG. B.2 – Les réponses des utilisateurs au questionnaire

	Croisé					Concentrique					Empilé				
	Niv. 1	**Niv. 2**	**Niv. 3**	**Niv. 4**	**Err (%)**	**Niv. 1**	**Niv. 2**	**Niv. 3**	**Niv. 4**	**Err (%)**	**Niv. 1**	**Niv. 2**	**Niv. 3**	**Niv. 4**	**Err (%)**
1-3-4-7	2.14	3.25	5.12	5.09	25	2.82	1.98	6.32	4.03	13	1.75	1.63	3.18	3.17	25
1-4-7-6	1.52	1.73	3.67	4.32	13	1.36	1.71	3.24	3.59	13	1.91	1.58	4.14	3.02	13
1-5-4-1	1.80	2.30	3.53	4.58	13	3.55	1.36	5.75	3.01	38	1.68	2.67	2.93	4.16	38
2-2-5-2	1.75	5.15	4.19	7.27	63	1.37	1.26	3.43	2.57	25	2.28	0.78	3.78	2.55	25
2-3-1-5	1.70	3.18	3.71	5.23	25	2.39	1.36	4.80	2.54	38	1.56	1.18	4.38	3.05	25
2-3-5-5	2.33	2.20	4.01	3.65	13	1.58	1.25	3.68	2.57	13	1.76	2.04	3.79	2.49	50
2-5-6-2	2.27	2.15	6.08	4.63	38	1.70	1.24	3.52	2.58	-	1.82	1.68	3.11	3.18	25
2-7-4-4	1.93	2.89	4.33	4.90	13	2.41	2.28	3.54	3.79	13	1.88	1.96	3.45	2.43	25
3-3-4-7	2.55	1.75	3.94	3.83	38	2.87	1.52	4.12	3.24	25	1.75	1.32	2.99	3.13	13
4-3-6-1	1.60	5.68	7.11	8.27	38	2.59	1.77	3.98	3.35	13	1.23	1.52	3.45	3.26	25
4-5-1-2	1.13	2.07	2.76	4.32	13	2.59	1.40	4.73	2.99	38	1.77	1.34	3.19	2.95	13
4-5-5-3	1.16	1.98	2.78	3.75	-	1.95	1.93	3.19	3.60	38	0.96	1.10	1.47	2.64	-
6-1-5-4	2.79	2.67	4.72	4.86	25	2.39	1.58	3.61	3.17	25	2.00	2.01	3.73	3.19	38
6-1-6-7	2.66	2.14	4.60	4.28	25	1.50	2.21	3.04	3.82	25	2.13	1.42	4.32	2.66	25
6-3-1-5	2.01	1.89	4.01	3.97	13	1.52	1.46	2.70	2.81	-	1.97	1.54	3.64	2.99	13
6-7-1-6	3.56	2.13	6.35	4.65	38	2.11	1.40	3.48	2.58	13	1.85	1.42	3.69	3.05	13
6-7-6-6	4.35	2.21	6.53	3.58	50	2.22	1.51	3.64	2.41	13	2.27	1.23	3.99	1.76	38
7-1-5-5	1.86	4.04	4.16	6.39	38	4.05	1.62	5.74	2.63	25	1.48	2.25	3.07	2.89	25
7-2-2-6	1.60	3.06	3.77	6.36	25	3.01	1.68	5.27	2.86	50	1.62	2.39	3.09	3.95	50
7-2-7-6	2.34	3.50	5.78	6.07	38	1.54	1.48	3.04	3.13	-	1.52	3.34	3.23	4.86	38
7-7-1-2	2.14	1.81	5.30	4.38	38	2.04	1.31	4.26	2.60	25	1.55	0.93	3.17	2.54	-

FIG. B.3 – Temps de sélection, par niveau hiérarchique

1	Chargez l'objet "Boite"
2	Agrandissez la table
3	Déplacez la table pour la centrer sur l'espace de travail
4	Nous allons maintenant créer un l'un des pieds de la table.
5	Créez une contrainte ponctuelle rectiligne
6	Sélectionnez la contrainte
7	Changez son volume d'influence en boite
8	Redimmensionnez le volume (environ 10 cm d'arête)
9	Changez la fonction d'extrusion en "porte"
10	Déplacez l'origine de la contrainte au bon endroit
11	Déplacez l'extrémité de la contrainte au bon endroit
12	Nous allons maintenant créer un second pied.
13	Créez une contrainte ponctuelle rectiligne
14	Sélectionnez la contrainte
15	Changez son volume d'influence en boite
16	Redimmensionnez le volume (environ 10 cm d'arête)
17	Changez la fonction d'extrusion en "porte"
18	Déplacez l'origine de la contrainte au bon endroit
19	Déplacez l'extrémité de la contrainte au bon endroit
20	Nous allons maintenant créer un troisième pied.
21	Créez une contrainte ponctuelle rectiligne
22	Sélectionnez la contrainte
23	Changez son volume d'influence en boite
24	Redimmensionnez le volume (environ 10 cm d'arête)
25	Changez la fonction d'extrusion en "porte"
26	Déplacez l'origine de la contrainte au bon endroit
27	Déplacez l'extrémité de la contrainte au bon endroit
28	Nous allons maintenant créer le dernier pied.
29	Créez une contrainte ponctuelle rectiligne
30	Sélectionnez la contrainte
31	Changez son volume d'influence en boite
32	Redimmensionnez le volume (environ 10 cm d'arête)
33	Changez la fonction d'extrusion en "porte"
34	Déplacez l'origine de la contrainte au bon endroit
35	Déplacez l'extrémité de la contrainte au bon endroit
36	Nous allons maintenant finaliser la table et la positionner dans la scène.
37	Appliquez la déformation à la table
38	Retournez la table, réduisez sa taille et déplacez-là légèrement vers la gauche.
39	Nous allons maintenant créer une sorte de cactus
40	Chargez l'objet "cactus"
41	Créez une contrainte ponctuelle rectiligne
42	Sélectionnez la contrainte
43	Déplacez l'origine de la contrainte
44	Redimmensionnez le volume (environ 20 cm de diamètre)
45	Changez la fonction d'extrusion en "lisse"
46	Changez le paramètre de la fonction d'extrusion à 0.5 environ
47	Appliquez la déformation
48	Nous allons maintenant ajouter une branche à notre cactus
49	Créez une contrainte ponctuelle curviligne
50	Sélectionnez la contrainte
51	Déplacez l'origine de la contrainte sur le morceau de cactus
52	Redimmensionnez le volume (environ 10 cm de diamètre)
53	Changez la fonction d'extrusion en "lisse"
54	Changez le paramètre de la fonction d'extrusion à 0.5 environ
55	Appliquez la déformation
56	Déplacez le cactus pour le placer sur la table et redimmensionnez-le
57	Duppliquez le cactus et placez-le à proximité de l'autre cactus, sur la table
58	Enfin, passez en affichage surfacique et admirez votre oeuvre !
59	La manipulation est terminée. Merci de votre participation !

FIG. B.4 – Actions de construction d'une scène avec DogmeRV

Table des figures

Liste des tableaux