N° d'ordre: 5265

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

M. Ludovic STERNBERGER

pour obtenir le grade de

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

Interaction en réalité virtuelle

---

soutenue publiquement le **28 novembre 2006**,
devant la commission d'examen composée de :

Mme Dominique BECHMANN, Directeure de Thèse,
Professeure à l'Université Louis Pasteur de Strasbourg I
M. Jean-Michel DISCHLER Rapporteur Interne,
Professeur à l'Université Louis Pasteur de Strasbourg I
M. Stéphane DONIKIAN Rapporteur Externe,
Chargé de recherches - HDR à l'IRISA de Rennes
M. Frédéric MERIENNE Rapporteur Externe,
Professeur à l'ENSAM de Chalon-sur-Saône
M. Jean-Pierre JESSEL Examinateur,
Professeur à l'Université Paul Sabatier de Toulouse

Table des matières

Introduction

Contexte - Domaine

Le sujet de ce mémoire se situe à l'intersection de deux domaines qui sont souvent associés : la *réalité virtuelle* et l'*interaction*.

La réalité virtuelle fait l'objet d'un intérêt croissant depuis une quarantaine d'années, et particulièrement depuis le début des années 1990. Cela pour au moins une raison. Elle permet de produire une réalité immatérielle présentée à l'esprit par l'intermédiaire des sens de l'être humain. Ce monde potentiel repose sur une série d'illusions coordonnées et réunies en un tout objectif pour l'esprit humain par l'ordinateur.

L'interaction homme-machine répond au besoin de faciliter la communication entre l'homme et la machine, et d'aider à la construction d'un modèle mental humain cohérent. Elle repose sur l'ensemble des moyens matériels et logiciels mis en œuvre pour favoriser l'interprétation des souhaits et réactions de l'être humain face à un ordinateur. Initialement, ce lien – on parlera encore d'interface – n'était que physique : l'utilisateur manipulait des cartes perforées sur les premiers ordinateurs. Par la suite, l'apparition des premiers écrans permit d'introduire une communication *via* le clavier et le texte, dématérialisant une partie de l'interaction. À la fin des années 70, l'apparition des premières interfaces graphiques $2D$ finit d'attester du rôle fondamental que joueraient désormais les aspects logiciels dans le dialogue entre l'homme et la machine.

Progressivement, l'apparition de la réalité virtuelle et du matériel approprié a permis de créer des environnements en 3 dimensions. Dès lors, l'interaction $2D$ devait laisser la place à une interaction dite $3D$, plus adaptée aux possibilités offertes par ces nouveaux environnements. Cependant, en comparaison des interfaces $2D$, dont les aspects matériels et logiciels sont désormais quasiment standardisés, le challenge est bien plus important dans un monde virtuel $3D$. L'interaction homme-machine en réalité virtuelle pose un certain nombre de problèmes inhérents à la fois au matériel qui est utilisé, et aussi à la mise au point d'outils virtuels adaptés à l'utilisation des périphériques.

Problématique générale

Alors qu'il aura fallu une bonne vingtaine d'années pour passer des premiers prototypes d'interfaces $2D$ aux environnements de bureau que l'on connaît et utilise aujourd'hui, la situation est bien différente en $3D$. La problématique de l'interaction en réalité virtuelle se pose sous deux optiques qui sont liées.

D'une part, le nombre de périphériques physiques est très important et ceux-ci sont souvent de nature très différente. D'une simple paire de lunettes à écrans de projection intégrés, l'utilisateur peut également être confronté à un environnement immersif dont la taille dépasse les $30m^3$. Il y a donc là une réelle nécessité de réussir à standardiser le matériel pour baisser les coûts, à limiter le nombre de périphériques utilisés et pérenniser les solutions existantes. Ce qui facilitera naturellement l'accès de la réalité virtuelle au plus grand nombre. D'un matériel trop expérimental, il est impératif d'aboutir rapidement à des périphériques ergonomiques, simples, fonctionnels et abordables.

D'autre part, l'utilisation de ce matériel dédié aux environnements virtuels n'est pas aisée pour le moment. De nombreuses recherches ont d'ores et déjà abouti à la création d'outils virtuels spécifiques à certaines tâches. Mais on peut constater que très peu d'applications de production existent à l'heure actuelle. Là encore, il est nécessaire de disposer de solutions logicielles standardisées, qui soient, dans la mesure du possible, indépendantes du matériel sous-jacent. Plus encore, en faisant un parallèle avec le monde de la $2D$, il faut disposer d'outils similaires permettant le développement rapide et efficace d'applications finalisées.

Position du problème à résoudre

Notre travail se situe donc dans l'optique de l'interaction logicielle en réalité virtuelle. Bien que l'étude des techniques du matériel existant soit un préalable indispensable à la compréhension du domaine, au même titre que la connaissance du fonctionnement des sens humains, nous nous consacrerons essentiellement aux aspects informatiques.

Objectifs

Notre objectif est d'améliorer l'interaction logicielle en réalité virtuelle.

Un état de l'art Nous souhaitons d'abord commencer par obtenir une vue globale et complète sur le domaine. Pour nous, cela signifie qu'il s'agit à la fois d'étudier les aspects logiciels et les aspects matériels de l'interaction en réalité virtuelle.

Une approche théorique simple de l'interaction Nous souhaitons profiter des connaissances que nous avons acquises pour proposer un guide de conception de l'interaction homme-machine, qui pourra être utilisé lors de la réflexion préalable à l'écriture d'un logiciel de réalité virtuelle.

Un ensemble d'outils adaptés à l'interaction Notre objectif final est de proposer au développeur un cadre de travail adapté à l'interaction en réalité virtuelle, par l'intermédiaire d'outils de développement puissants et simples à mettre en œuvre.

Démarche

Le premier objectif est de réaliser un état de l'art très fouillé sur l'interaction en réalité virtuelle. Sur ce point, notre démarche consiste à couvrir de la façon la plus large possible le domaine, sans se perdre dans ses méandres. Nous proposons de couvrir à la fois les aspects logiciels et les aspects matériels de l'interaction. Mais avant tout, il nous faut poser plus précisément les définitions des termes et des notions fondamentales qui seront utilisées dans la suite du manuscrit.

Le second objectif est de proposer une nouvelle approche de l'interaction adaptée au développement d'applications et de techniques d'interaction de réalité virtuelle. Cela signifie établir des règles de conception des techniques d'interaction, et de l'ensemble des objets virtuels que ces derniers manipulent. C'est en considérant la chaîne d'interaction dans sa globalité qu'il est possible de déterminer ces règles, et d'autres lois.

Les deux premiers objectifs mènent au troisième qui est de proposer au développeur un ensemble d'outils de développement d'applications de réalité virtuelle. En ayant examiné préalablement toute la chaîne d'interaction, et en considérant que les outils doivent être les plus intuitifs possibles pour l'utilisateur final, nous souhaitons nous reposer sur la reconnaissance de postures et de gestes. Notre démarche sera de proposer des outils et des objets qui font exactement ce qu'on leur demande, c'est-à-dire ni trop, ni trop peu. Cette boîte à outils constitue la partie concrète de ce travail de thèse.

Enfin, il semble important de tester et confronter notre travail à la réalité de la conception logicielle. Cela signifie qu'il nous faut tester les concepts logiciels, eux-mêmes issus de notre travail théorique préalable, dans des applications suffisamment complexes pour nous assurer que notre démarche est correcte.

Apports

Nos apports à l'interaction logicielle en réalité virtuelle portent principalement sur trois aspects.

Premièrement, nous proposons un état de l'art complet sur l'interaction notamment avec les techniques récentes, en y adjoignant notamment un nouveau chapitre sur l'entrée de symboles. Notre souhait est d'inscrire ce travail, de manière claire et précise, dans le triptyque de l'interaction : le matériel, le logiciel et les sens de l'homme que l'on cherche à stimuler.

Deuxièmement, l'élaboration d'une théorie de l'interaction conduite par la dimension fournit à l'interaction des règles et des lois à observer lors de la phase de conception d'une application virtuelle, et ce, quel que soit le matériel utilisé.

Les notions de dimensionnalité visuelle et fonctionnelle permettent d'avoir une approche plus globale et systématique, peu importe le cas de figure, et surtout de mettre en œuvre rapidement son application au sein d'un canevas de travail.

Troisièmement, la conception d'une librairie de développement rapide et efficace d'applications de réalité virtuelle tend à apporter au domaine ce qui lui manque depuis toujours : une abstraction efficace du matériel, tout en proposant une interface de développement simple. Une librairie standard permet à la fois d'offrir un cadre propice au développement de nouvelles techniques d'interaction, et autorise la comparaison des techniques entre elles. En définitive, un tel outil de travail évite de réécrire complètement une application dédiée à une plateforme matérielle spécifique. Sobrement nommée vrLib, cette librairie permet tout aussi bien de porter une application $2D$ en environnement $3D$, que d'utiliser et/ou concevoir des outils d'interaction plus évolués.

Finalement, dans le cadre de notre démarche, nous choisissons deux applications pour tester les concepts théoriques implantés dans notre librairie d'interaction. Nous avons développé d'autres applications qui nous permettent de voir quel est le degré d'efficacité de notre trousse à outils, puis d'en affiner les réglages.

Structure du mémoire

La structure du mémoire s'organise de la façon suivante. Nous avons choisi de scinder ce mémoire en deux parties : l'une consacrée à un état de l'art sur l'interaction en réalité virtuelle, et l'autre sur nos travaux. Chaque partie est composée de trois chapitres distincts.

Le chapitre 1 introduit le domaine de la réalité virtuelle et des notions qui y sont associées. En particulier, nous montrons comment les sens de l'homme construisent un modèle mental cohérent de l'environnement dans lequel il est plongé. Le chapitre 2 cherche à couvrir de manière aussi exhaustive que possible les méthodes d'interaction $3D$ dédiées à la réalité virtuelle. Le chapitre 3 complète cet état de l'art en abordant les aspects matériels rencontrés. Le chapitre 1 de la partie 2 présente les grandes lignes de la vrLib, la librairie d'interaction que nous avons développée. Le chapitre 2 de la partie 2 expose notre théorie de l'interaction dirigée par la dimension, ainsi que notre travail sur l'utilisation des mains pour la reconnaissance de postures et de gestes. Le chapitre 3 de la partie 2 vient compléter la seconde partie de ce manuscrit en présentant plusieurs applications construites autour de cette librairie.

Première partie

État de l'art

Chapitre 1

La réalité virtuelle

1.1 Historique

La réalité virtuelle semble être aujourd'hui une technologie de pointe. Pourtant lorsque Jaron Lanier invente le terme de « réalité virtuelle » en 1986 cette science n'est pas nouvelle. Elle trouve ses racines dans la première moitié du 19^{e} siècle avec l'invention d'un appareil capable de reproduire une image en 3 dimensions.

La réalité virtuelle est apparue sous différentes formes. Bien conscients de l'importance que jouent les sens dans la vie de l'homme, certains artistes et scientifiques ont rapidement cherché à confronter l'être humain à des univers auxquels il n'avait pas accès dans le monde réel.

Depuis l'Antiquité, de nombreux artistes ont essayé de « courber » les images de la réalité en peignant des trompes-l'œil, car ils avaient compris, du moins saisi, que les sensations que nous éprouvons peuvent être altérées presque à volonté. Notre cerveau interprète ce que nous capturons de la réalité. Pour deux personnes qui sentent la même odeur, les impressions ressenties ne sont pas identiques.

Le sens le plus touché par cet état de fait est la vue. En 1824, Peter Roget découvre le principe de la persistance rétinienne et quelques années plus tard, en 1832, les premières images stéréoscopiques font leur apparition grâce au stéréoscope de Charles Wheatstone. La figure 1.1 montre un stéréoscope portatif de A.H. Baird, le Lothian, datant de 1895 et reposant sur deux prismes lenticulaires qui dirigent le regard de chaque œil sur une image différente. Puis, dès le début du 20^{e} siècle, les premiers pilotes d'avions issus des écoles de pilotage font leurs classes sur ce qui peut être qualifié de précurseurs aux simulateurs de vol. Pour recréer les conditions de vol, la simulation du pilotage et l'interaction avec les différents organes de l'avion sont obtenues à l'aide d'un équipement rudimentaire constitué de chaises, de poulies et de manches à balai.

Les images d'animations et de films, modifiables à souhait, ont été utilisées dès les débuts du cinéma pour recréer artificiellement des conditions d'ambiances

Fig. 1.1: Le stéréoscope portatif à deux lentilles de A.H. Baird (1895).

et d'atmosphères tout à fait particulières. L'informatique a fait irruption beaucoup plus tardivement fait irruption dans le traitement de l'image ; mais en l'espace de quelques décennies seulement, les images numériques ont pris une place essentielle dans la production d'œuvres cinématographiques.

En 1962 Morton Heilig invente le Sensorama, qui permettait de simuler le trajet d'un utilisateur dans une voiture. Des images et des sons, préalablement enregistrés, étaient projetés devant le pare-brise d'une voiture dans laquelle le spectateur prenait place. Le spectateur pouvaient sentir certaines odeurs et vibrations qui complétaient la sensation d'immersion. En 1966, Ivan Sutherland désire pousser plus avant le concept de « contact intime » entre l'homme et la machine, en plaçant un utilisateur à l'intérieur d'un monde en trois dimensions engendré par ordinateur. C'est ainsi qu'il imagine et crée le premier casque de réalité virtuelle monoscopique.

Il faudra attendre 1970 pour que Daniel Vickers améliore le système de Sutherland en ajoutant la stéréoscopie. L'utilisateur dispose alors d'un écran par œil et c'est le cerveau qui recompose l'image pour former mentalement une vue stéréoscopique. Il équipe également son casque d'un capteur de position et d'orientation de la tête, afin que chaque déplacement entraîne un nouveau calcul du point de vue.

La vision ne constitue pas, à cette époque, le seul centre d'intérêt des chercheurs dans la voie des sensations artificielles. En 1971, le premier prototype d'interface à retour d'effort (projet GROPE-1) est réalisé à l'Université de Caroline aux États-Unis. Ce projet permet de contraindre les mouvements dans un champ de forces électrostatiques $2D$. Plus tard, d'autres systèmes font leur apparition, comme par exemple le système des laboratoires ICA-ACROE en France et le LEPES, avec le nanomanipulateur qui permet de sentir le contact des objets d'échelle nanométrique observés à l'aide d'un microscope à force atomique.

Le début des années 80 marque un intérêt grandissant du tout public et des entreprises pour l'informatique. Les ordinateurs s'invitent dans les foyers avec l'Apple II de Macintosh et le PC d'IBM. Les pilotes de l'armée de l'air apprennent à piloter sur des simulateurs de vol entièrement contrôlés par ordinateur. En 1981, le premier gant de données est inventé par Thomas Zimmerman.

Cette nouvelle interface permet à l'utilisateur de communiquer avec la machine par l'intermédiaire de ses mains.

En 1985, la NASA combine gants de données et casque de vision stéréoscopique pour entraîner ses futurs astronautes aux prochaines missions spatiales. La répétition de leurs mouvements et les entraînements face aux situations d'échec leur permet de se préparer le mieux possible aux problèmes qui pourraient survenir dans l'espace. Dès 1986, l'agence spatiale américaine teste des techniques d'interaction virtuelles de vol, de mise à l'échelle et de rotation, toujours à l'aide d'un casque et de gants.

En 1986, Jaron Lanier introduit le terme de « *virtual reality* ». Il concevait alors la réalité virtuelle comme une réalité synthétisée, perçue par nos sens, et avec laquelle nous pouvons interagir *via* l'outil informatique.

Après l'invention des casques de réalité virtuelle, limités à une utilisation individuelle, des salles immersives à destination de plusieurs utilisateurs simultanés sont imaginées. En 1992, Tom De Fanti de l'Université d'Illinois invente le système CAVE. Ce système permet à plusieurs utilisateurs d'être plongés ensemble dans un environnement virtuel à taille humaine.

Depuis la fin des années 90, les interfaces de réalité virtuelle (casques de stéréovision, traqueurs, interfaces haptiques et murs immersifs) se sont généralisées dans les universités et les entreprises. Pourtant, le chemin sera encore long avant de pouvoir disposer d'interfaces standardisées reposant sur un maximum de sens et accessibles au grand public pour un coût raisonnable.

1.2 Plusieurs définitions

Les termes de Réalité Virtuelle et Environnements Virtuels sont les deux termes les plus populaires et les plus souvent rencontrés dans la littérature, mais il y en a bien d'autres. Ainsi, pour désigner la réalité virtuelle, il n'est pas rare d'utiliser les termes de Mondes Artificiels, Mondes Virtuels, Réalité Artificielle, ou encore d'Expérience Synthétique. Bien que toutes ces expressions désignent le même domaine, la définition n'est pas fixée ; chacun peut y définir ses propres limites.

Ainsi la réalité virtuelle n'a pour le moment pas de définition unique. En effet, le terme couvre des domaines très différents selon que celui qui l'utilise travaille sur la synthèse d'images stéréoscopiques ou bien sur la reproduction d'odeurs. Ainsi, chaque utilisateur donne sa propre définition, en fonction des intérêts qu'il y associe.

Généralement, on parle de réalité virtuelle pour décrire un monde synthétique où l'on peut éprouver des sensations de même nature que dans la réalité : voir, toucher, manipuler différents objets. Ainsi, la réalité virtuelle ne s'arrête pas simplement à la seule synthèse réaliste d'images stéréoscopiques. Tous les sens peuvent être mis à contribution pour plonger un peu plus l'utilisateur dans un univers virtuel. Pour avoir la sensation que cet environnement puisse exister

dans son quotidien, l'utilisateur doit devenir une partie active intégrante de la scène immatérielle qui se joue devant ses yeux.

Une définition générale a été élaborée par Aukstakalnis et Blatner [AB92]. Pour eux, *la réalité virtuelle est un moyen pour les humains de visualiser, manipuler et interagir avec des ordinateurs et des données extrêmement complexes.*

La réalité virtuelle constitue une réalité plausible mais qui n'existe aucunement. Il ne s'agit que d'une illusion que l'on cherche à parfaire en utilisant le maximum de sens de l'utilisateur. Ceux-ci constituent d'ordinaire le lien entre l'esprit et le monde réel, le seul moyen de communication entre les deux. Il est donc possible de remplacer les données perçues du réel par des données construites artificiellement, de manière à ce que le monde artificiel paraisse tout aussi envisageable que le monde réel.

Pour Gigante *et al.* [EGJ93], la réalité virtuelle est *l'illusion de la participation à un environnement synthétique plus que l'observation extérieure d'un environnement. La réalité virtuelle repose sur une troisième dimension, des dispositifs stéréoscopiques portés sur la tête, de membres du corps suivis dans l'espace et d'un environnement sonore. La réalité virtuelle est une immersion, une expérience multi-sensorielle.*

Ainsi, pour créer l'illusion de la réalité, l'utilisateur dispose d'*interfaces sensorielles* et *motrices* qui lui permettent de restituer des sensations et de reconnaître ses actions physiques dans le monde virtuel. Nous verrons un peu plus tard que la vue est le principal canal d'interaction avec le monde extérieur. C'est le sens qui participe pour près de 70% à l'illusion. En outre, l'utilisateur ne doit pas simplement être confronté à un environnement virtuel en tant que simple spectateur, mais avant tout en tant qu'acteur.

Pour Von Schweber [VS95], *la réalité virtuelle permet de naviguer et voir un monde en trois dimensions en temps réel, avec six degrés de liberté. [...] Avant tout, la réalité virtuelle est un clone de la réalité physique.*

Le monde virtuel dans lequel se promène l'utilisateur ne doit pas être uniquement réaliste du point de vue de la synthèse d'images, il doit avant tout paraître crédible. Par exemple, il est bien d'avoir une poignée de porte virtuelle bien détaillée et rendue, mais il est certainement plus important que la main virtuelle ne traverse pas la poignée et reste au contraire bien accrochée.

Slater *et al.* [SU94] pensent que la réalité virtuelle est une réalité réelle ou simulée dans laquelle le Moi croit qu'il est dans un environnement autre que celui dans lequel se trouve le corps physique. Le *Moi perçoit l'information sensorielle comme étant corrélée avec une réaction proprioceptivement valide de son corps (comportement et état) dans cet environnement.*

Fuchs [Fuc03] nous donne une définition assez intéressante et globale de ce que représente, pour lui, la réalité virtuelle. *La réalité virtuelle est un domaine scientifique et technique exploitant l'informatique et des interfaces comportementales en vue de simuler dans un monde virtuel le comportement d'entités*

3D, qui sont en interaction en temps réel entre elles et avec un ou des utilisateurs en immersion pseudo-naturelle par l'intermédiaire de canaux sensori-moteurs. C'est cette dernière définition que nous retiendrons, car c'est celle qui nous semble la plus complète et situant le mieux notre problématique sur l'interaction.

Nous venons de voir que pour être viable, la réalité virtuelle doit être crédible pour renforcer la sensation d'immersion de l'utilisateur dans un environnement *3D*. Elle doit également fournir des mécanismes d'interaction originaux, tout en restant simples et intuitifs d'utilisation. Enfin l'interaction ne doit pas s'arrêter au seul sens de la vision, mais au contraire s'étendre au maximum de sens humains. Nous allons nous intéresser un peu plus à la multi-sensorialité et à la sensation d'immersion.

1.2.1 Immersion, présence et mal virtuel

Qu'est-ce que l'immersion ? Dans Le Petit Robert on trouve la notion « d'absorption mentale ». Nous pouvons voir l'immersion comme l'état d'esprit d'une personne qui exclut le monde qui l'entoure et qui est totalement focalisée par un environnement imaginaire ou artificiel. Cet état peut être plus ou moins marqué. Cela se produit par exemple lorsqu'on lit un livre et qu'on se retrouve plongé dans l'histoire, à tel point que lorsqu'une personne nous parle nous ne nous en rendons pas compte. L'immersion est un terme utilisé pour décrire le sentiment d'appartenance à un environnement crédible, différent de l'environnement réel. Elle peut avoir lieu à partir d'un livre ou d'un film [Hac03].

En réalité virtuelle l'immersion permet de mesurer le degré de réalisme créé par un système de réalité virtuelle. Rheingold [Rhe91] définit l'immersion comme étant *l'illusion d'être dans une scène générée par ordinateur. Elle peut être totale ou partielle selon les périphériques d'interaction utilisés.* D'ailleurs, la stéréoscopie et l'interactivité en temps réel jouent un rôle déterminant à cette sensation d'immersion, car elles placent l'utilisateur au centre de l'histoire qui se déroule devant ses yeux : il en devient l'acteur principal.

Certains auteurs utilisent le terme « présence » à la place du mot immersion. Pour Bowman *et al.* [BKH97], la présence se ramène à la sensation d'immersion de l'utilisateur, c'est-à-dire la sensation qu'il a d'être dans l'environnement. Pour Slater *et al.* [SU94] [SUS95], l'immersion est une description d'une technologie, qui peut être réalisée à des degrés variables. Dans un environnement virtuel, il y a une condition nécessaire pour parler d'immersion : le système doit maintenir en permanence au moins une modalité sensorielle (en général la vision). Le degré d'immersion augmente par exemple en améliorant la précision des capteurs de position et d'orientation, en disposant d'une représentation du corps (avatar) plus riche et plus détaillée, en diminuant le temps de latence, etc. Finalement, l'immersion peut mener à la sensation de présence. Il s'agit d'une propriété psychologique émergeant d'un système virtuel, qui se rapporte à la sensation d'être là (c'est-à-dire dedans), dans le monde créé par ordinateur. L'immersion est une condition nécessaire plutôt que suffisante à la sensation de présence, alors que la présence décrit un état de conscience associé à la technologie virtuelle. Ici, l'idée fondamentale est que les techniques d'interaction qui maximisent la cor-

respondance entre données proprioceptives et données sensorielles, maximisent la sensation de présence. En outre, lorsque l'utilisateur dispose d'un avatar de son propre corps, la sensation de présence s'en trouve améliorée.

Whitton [Whi03] ajoute que le degré d'immersion dépend du nombre de sens qui sont simulés et stimulés par le système de réalité virtuelle et à quel niveau les utilisateurs sont isolés de l'environnement du monde réel. La qualité d'immersion varie avec la fidélité des simulations physiques [Lin03], du rendu (pour tous les sens), et de la représentation (c'est-à-dire l'affichage) des données. Ainsi, les stimuli sensoriels doivent être consistants et synchronisés pour que l'utilisateur perçoive le monde qu'ils décrivent comme cohérent et prévisible. On parle de consistance à travers les sens.

Les notions d'immersion et de sensation de présence sont à prendre en compte lors de la conception d'une interface de communication entre l'homme et la machine, surtout si l'on cherche à rapprocher ces derniers. En effet, lorsque l'immersion n'est pas très bonne, des effets secondaires peuvent apparaître, comme par exemple des sensations de nausée. Il s'agit du mal virtuel ou mal du simulateur[1], qui est directement dépendant de la sensation de présence de l'utilisateur dans l'univers virtuel. La sensation de présence et le mal du simulateur sont corrélés négativement : lorsque l'un des deux diminue, l'autre augmente.

La sensation de présence est une perception difficile à quantifier. Certains auteurs ont cherché à mesurer son importance en observant le rythme cardiaque et les variations de température corporelle d'un sujet placé au centre d'une scène virtuelle stressante.

Par exemple, Meehan *et al.* [MRWBJ03] se sont intéressés aux effets de la fréquence de rafraîchissement des images sur la sensation de présence en plaçant des utilisateurs aux dessus d'un vide simulé. Ils partent de l'hypothèse que plus la sensation de présence est importante et plus le stress (et du même coup le nombre de pulsations cardiaques par seconde) augmente. Ils remarquent qu'il ne semble pas y avoir d'effets significatifs de la latence sur la sensation de présence, et que la peur de tomber ne dépend pas de ce paramètre d'affichage.

Lin *et al.* [LDP+02] ont cherché à déterminer les effets du champ de vision sur la présence. La sensation de présence semble augmenter lorsque le champ de vision est plus grand, alors que dans le même temps, le mal virtuel diminue. On peut émettre l'hypothèse qu'en élargissant le champ visuel, il y a moins d'informations visuelles réelles qui viennent « contredire », en quelque sorte, les informations visuelles virtuelles. Le cerveau est moins désorienté dans la compréhension de ce qu'il interprète, puisqu'il y a moins d'informations visuelles contradictoires.

LaViola *et al.* [LJ00] se sont intéressés aux facteurs qui pourraient être en cause dans ce mal. Ils s'intéressent en particulier au système vestibulaire[2], qui fournit les informations du mouvement et de l'orientation de la tête dans l'es-

[1] a. *cybersickness*

[2] partie de l'oreille interne logée dans l'os temporal.

pace, ainsi qu'au système visuel. Plusieurs théories pourraient expliquer l'origine du mal :

- La théorie du conflit sensoriel est celle qui est la plus en vogue aujourd'hui. Le mal se produit lorsque plusieurs sens renvoient des informations contradictoires les uns par rapport aux autres.
- La théorie de l'instabilité posturale qui dit que la priorité du corps humain est d'assurer sa propre stabilité dans l'espace. La perception des mouvements incontrôlés seraient minimisés. Mais lorsque la période d'instabilité est longue, le mal virtuel devient important et durable.

Ils citent également les problèmes de matériel comme cause probable du mal virtuel : erreurs des capteurs de position et d'orientation, temps de latence entre deux images affichées, clignotement qui entraîne la fatigue des yeux, etc. Plusieurs solutions sont envisagées pour diminuer les symptômes : placer l'utilisateur sur une plateforme physique pour la navigation, stimuler directement le système vestibulaire, montrer à l'utilisateur un plus grand nombre d'images *rest frames*[3] qui permettent de mieux percevoir le mouvement, ou encore augmenter son expérience des environnements virtuels.

Présence, immersion et mal virtuel sont des aspects de la réalité virtuelle qui ne doivent pas être négligés lors de la conception d'un nouvel environnement. La qualité des périphériques et de la conception du monde sont à prendre en considération en priorité. On pourra trouver plus d'informations sur la présence et l'immersion dans un état de l'art particulièrement complet sur ces deux notions de [SVDSKVDM01].

1.2.2 Autres définitions

Voici quelques notions supplémentaires que nous utiliserons dans la suite de ce manuscrit. Elles sont importantes et apparaissent dans la plupart des ouvrages de référence.

Virtualisation Ellis [Ell91] définit la virtualisation comme le processus par lequel un être humain interprète une sensation (issue d'un ou plusieurs de ses sens) reproduite pour être étendue dans un environnement autre que celui dans lequel il existe physiquement.

Environnement virtuel Un environnement virtuel peut être vu comme un modèle $3D$ de données réelles ou imaginaires que l'on peut visualiser et avec lesquelles on peut interagir en temps-réel [Hac03]. Des informations sensorielles complémentaires (sonores, tactiles, ...) peuvent venir enrichir ce modèle. Pour Slater *et al.* [SU94], cet environnement virtuel devient immersif lorsque la représentation virtuelle du corps de l'utilisateur[4] (tout ou partie) est affichée dans l'environnement virtuel, et qu'il peut réaliser un certain nombre d'actions sur cet environnement (à l'aide de capteurs de position et d'orientation, de gants de données, ...).

[3] c'est ce qui reste d'une image lorsqu'on retire de celle-ci l'objet d'intérêt.
[4] on parle alors d'avatar virtuel

Interaction L'interaction correspond à l'ensemble des influences réciproques entre l'homme et l'ordinateur par l'intermédiaire d'interfaces sensorielles, d'interfaces motrices, et de techniques d'interaction. Il existe deux types d'interaction : celle de tous les jours issue du réel, et celle dite *imaginaire* qui est impossible dans le monde réel.

Contact virtuel On parle également de contact virtuel [Lin03] lorsque l'utilisateur est en contact avec des objets du monde virtuel, soit directement, soit indirectement. Cette notion peut parfois être rencontrée en synonyme au terme interaction, défini au-dessus.

Technique d'interaction Pour interagir, les utilisateurs ont besoin de *méthodes permettant [...] d'effectuer une tâche d'interaction dans un environnement virtuel* [Hac03]. Une technique d'interaction décrit la manière de se servir d'un périphérique pour accomplir une tâche sur un ordinateur [FVDFH90] ; elle constitue la couche logicielle et désigne la méthode ou le scénario d'utilisation de l'interface motrice utilisée dans l'application. En réalité virtuelle, on parle d'**interaction** $3D$ pour dire que l'on interagit avec des objets virtuels dans un espace tridimensionnel. Le terme méthode d'interaction peut être compris comme un synonyme à technique d'interaction.

Paradigme d'interaction Certains auteurs utilisent la notion de paradigme d'interaction pour parler d'un regroupement de techniques d'interaction. Dans la suite, nous considérerons un paradigme comme une technique d'interaction, sans faire de distinction particulière.

Métaphore d'interaction Il n'est pas rare que le terme de métaphore d'interaction soit assimilé à celui de technique ou de paradigme, alors qu'il y a une différence de niveau conceptuel importante entre ces deux notions. On parlera de métaphores pour les analogies utilisées par les fonctions de transfert. Une métaphore d'interaction signifie qu'un outil virtuel est la métaphore, la transposition, d'un objet ou d'un concept réel.

Périphérique d'entrée Un périphérique d'entrée est un dispositif physique contrôlé par l'utilisateur, qui fournit des informations à l'ordinateur. Ceci inclut les périphériques comme la souris et le clavier. On peut trouver comme synonymes les termes de dispositif d'entrée, de périphérique de saisie, ou encore d'interface d'entrée.

Périphérique de sortie Un périphérique de sortie est un dispositif physique utilisé pour présenter des informations générées par un ordinateur à l'utilisateur, quelle que soit leur nature (sons, images, odeurs, ...). Un écran d'ordinateur est un exemple de périphérique de sortie. On peut trouver comme synonymes les termes de dispositif de sortie, et d'interface de sortie.

Périphérique isotonique Un périphérique isotonique est un dispositif sur lequel la tension musculaire appliquée est constante, comme par exemple la souris. Ce type de matériel peut être dédié au contrôle de la position, comme les

capteurs de position qui bougent librement pendant les mouvements de l'utilisateur.

Périphérique isométrique Un périphérique isométrique est un dispositif élastique sur lequel la tension musculaire varie en fonction du déplacement, comme par exemple un joystick. Ce type de matériel peut être dédié au contrôle de la vitesse puisque l'accélération dépend directement de la tension musculaire appliquée.

Traqueur Un traqueur[5] est un capteur de suivi de mouvements qui retourne des informations sur sa position et son orientation dans l'espace. En combinant plusieurs traqueurs de concert, il est possible de suivre les différents mouvements d'un utilisateur, et ainsi d'adapter le calcul des images selon son propre point de vue. On parle alors de suivi de mouvement[6], en précisant parfois la partie du corps qui est repérée (la tête, les mains, le torse, ...).

1.3 Les sens de l'homme rendent réel le virtuel

Nous avons vu dans la section 1.2.1 que, pour que le sentiment d'immersion soit de qualité, il est nécessaire que le nombre de sens stimulés soit le plus grand possible. Morton Heilig, considéré dans de nombreux ouvrages comme le père de la réalité virtuelle, est l'inventeur du Sensorama dans les années 1950. Le propos de ce dispositif était de donner, à ceux qui assistaient aux représentations, l'illusion complète de la réalité, par l'intermédiaire d'un écran très large, de sons stéréophoniques pré-enregistrés, d'odeurs vaporisés dans l'air, de variations de température, et de mouvements du sol ; le Sensorama simulait une balade à motocyclette dans les rues de New York. Selon sa classification, les sens d'un utilisateur monopolisent plus ou moins son attention. La vision participe pour 70%, l'ouïe pour 20%, l'odorat pour 5%, le toucher pour 4%, le goût pour 1%. La vision est de loin le sens prépondérant ; c'est donc celui qui sera stimulé en priorité pour convaincre notre cerveau qu'il interagit avec un environnement virtuel.

1.3.1 La vision

La vue, l'ouïe, le goût, le toucher et l'odorat sont les cinq sens qui permettent à l'être humain de saisir des informations du monde extérieur. La vision est le sens le plus développé, du moins celui qui apporte le plus d'informations sur notre environnement. On peut chiffrer la vitesse de transmission des informations d'un sens en évaluant le nombre de bits par seconde qu'il envoie au cerveau. La vision a une vitesse de 4,3 millions de bits par seconde, alors que l'ouïe ne fournit que 8000 bits par seconde *via* les oreilles. La vision occupe ainsi 70% de l'activité du cerveau consacrée aux différents traitements sensoriels.

La vision a donc été le premier des sens à être étudié et utilisé en réalité virtuelle, et notamment grâce à la possibilité de fournir des images stéréoscopiques à notre cerveau. Plus précisément, comme nous le verrons plus tard, il

[5] a. *tracker*
[6] a. *tracking*

suffit d'afficher une image légèrement différente devant chaque œil pour que le cerveau en interprète une image stéréoscopique ; exactement comme il le fait en permanence dans le monde réel.

Le fonctionnement de l'œil est assez complexe. Les rayons lumineux parviennent sur la rétine qui est tapissée de capteurs, qui transforment l'énergie lumineuse en impulsions électriques. Ces cellules photoréceptrices spécialisées sont divisées en deux groupes : les cônes dédiés à la vision photonique et les bâtonnets consacrés à la vision scotopique. Les cellules réceptrices transmettent l'information électrique jusqu'au lobe occipital, situé à l'arrière du cerveau pour produire la vision. L'image de chaque œil est ensuite combinée pour produire une vision stéréoscopique, c'est-à-dire une perception tridimensionnelle de l'environnement.

Vision stéréoscopique La stéréovision, qui semble être un processus tout à fait anodin résulte en fait d'une période d'apprentissage durant les premiers mois de la vie par synthèse des sensations tactiles, auditives et visuelles. Lorsque l'on regarde un objet, il se forme une image sur le fond de la rétine de chaque œil. La perception binoculaire de la profondeur et de la distance d'un objet résultent du chevauchement des champs visuels des deux yeux. En réalité virtuelle, la vision stéréoscopique est réalisée en calculant deux images légèrement différentes, une pour chaque œil. Pour séparer les deux images, on utilise essentiellement deux techniques : la stéréoscopie passive et la stéréoscopie active.

La stéréoscopie passive Le stéréoscopie passive reposer sur l'utilisation de filtres polarisants. Deux vidéo-projecteurs émettent sur une seule surface et en même temps une image pour chacun des yeux. Chaque faisceau lumineux est polarisé de façon différente. L'utilisateur porte des lunettes qui séparent les deux faisceau selon leur polarisation, selon deux technologies : linéaire ou circulaire. Bien que cette solution nécessite de doubler le matériel nécessaire, elle est simple à mettre en œuvre. Son inconvénient majeur provient de l'orientation et de la position de l'utilisateur par rapport à la surface d'affichage. Par exemple, lorsque l'observateur penche trop la tête avec une technologie à filtres polarisants linéaires, il peut apparaître des interférences entre les deux faisceaux lumineux.

La stéréoscopie active La stéréoscopie active est plus compliquée à réaliser. Elle repose sur une alternance de l'affichage des images pour l'œil gauche et l'œil doit, synchronisé avec des lunettes à obturateur que porte l'utilisateur. Citons deux exemples de configurations technologiques.

Avec les casques de réalité virtuelle[7], les deux images sont affichées sur deux petits moniteurs à cristaux liquides montés en face des yeux. Chaque œil reçoit l'information nécessaire à la fusion des deux images, ce qui permet au cerveau de pouvoir interpréter correctement la profondeur.

Lorsque l'on utilise un dispositif rétroprojeté comme le *Workbench*, il est impossible d'afficher simultanément une image par œil. Heureusement, il est possible de tromper notre cerveau en profitant de nos capacités visuelles quelque

[7] a. *HMD* pour *Head Mounted Display*

peu limitées. En pratique, on place un obturateur devant chaque œil et on affiche – successivement et rapidement – sur un écran (il faut que l'écran puisse être rafraîchi au moins à 60Hz) deux images. Chaque image est affichée le temps que l'obturateur soit fermé devant un œil, puis c'est l'obturateur de l'autre œil qui est fermé et une autre image affichée. La persistance rétinienne permet de « bluffer » le cerveau, alors que pendant ce temps, une autre image est affichée sur l'autre œil. La figure 1.2 donne le fonctionnement schématique du principe de fusion mentale opéré par le cerveau humain pour percevoir la distance et la profondeur.

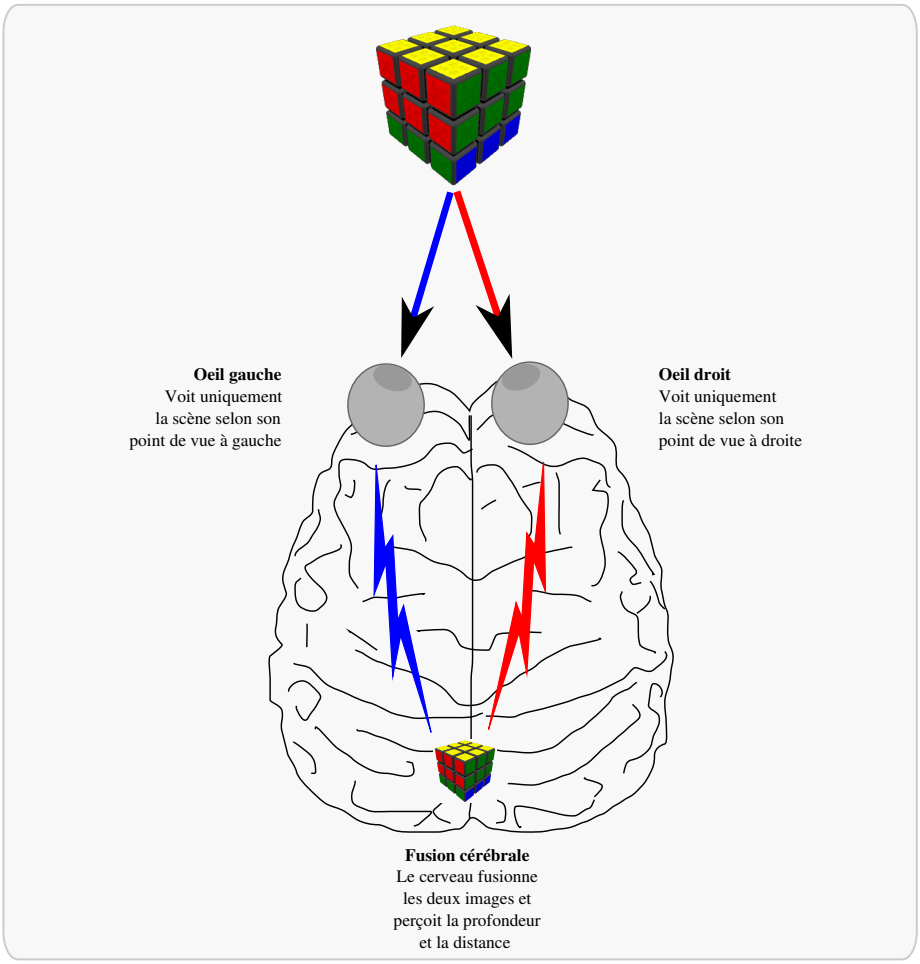


Fig. 1.2: Fusion cérébrale menant à la vision stéréoscopique.

Plusieurs problèmes peuvent se poser. En effet, chaque individu a un écartement des yeux différent, or les images sont souvent calculées à partir d'une valeur d'écartement moyen – environ 7cm. Du coup, et puisqu'il est difficile de calibrer le dispositif de stéréovision, les utilisateurs peuvent éprouver des gènes (désorientation, nausées, ...). Un autre problème concerne la distance d'accommodation des yeux. Dans le cas d'un *HMD*, les yeux accommodent sur chaque

écran à cristaux liquides, et sur un plan de travail comme le *Workbench*, ils accommodent sur les écrans de projection, alors que le système donne l'illusion de se trouver plus près ou plus loin. Ce décalage entre accommodation et convergence – c'est-à-dire la distance supposée entre les objets virtuels et l'utilisateur – est également à l'origine de phénomènes de désorientation et de malaise.

Disposer d'une vision binoculaire ne sert à rien si on ne l'exploite pas. L'utilisateur doit pouvoir tourner autour d'un objet, voir dessous, dessus et s'approcher ou s'éloigner. La position de ses yeux doit donc être connue du système informatique pour calculer l'image monoculaire projetée à chaque œil. On utilise des traqueurs ou des marqueurs pour suivre les mouvements de la tête, et deux nouvelles images sont calculées en permanence, puisque le corps est sans cesse en mouvement.

1.3.2 Le toucher

La vision n'est pas le seul sens à être virtualisé. Bien que le toucher n'entre que pour 4% de la monopolisation de l'attention d'un utilisateur, il a focalisé de nombreux centres de recherche. Les périphériques qui permettent de stimuler le toucher, et en particulier les interfaces haptiques[8], sont utilisés dans des domaines manuels comme la chirurgie , où les praticiens peuvent s'entraîner virtuellement avant de faire réellement les opérations, et la CAO où les ingénieurs peuvent tester le bon fonctionnement des pièces avant leur usinage.

Le sens du toucher est dû à la présence de nombreux récepteurs et corpuscules situés sous la peau. Chacun joue un rôle bien déterminé : sensation de froid, de chaleur, de douleur ou de pression. *In utero*, le toucher est le premier sens à faire son apparition. Il est principalement situé dans les zones du visage, des mains et des pieds.

En réalité virtuelle, on utilise des interfaces haptiques[9] pour interagir avec des objets virtuels. Il est possible de les saisir, les soulever, sentir leur rugosité. Ces dispositifs reproduisent les perceptions du toucher et le retour de force (ou retour d'effort) exercé par un objet sur l'utilisateur (rétroaction haptique).

Les interfaces haptiques offrent la possibilité de rendre un environnement virtuel plus cohérent. En effet, l'utilisateur préférera que sa main soit arrêtée par un objet virtuel plutôt qu'elle le traverse. *A priori* lorsqu'il pousse un objet posé sur une table, il aimerait sentir les frottements entre la table et l'objet. Ces sensations découlent de phénomènes physiques qui sont modélisés en informatique grâce à la gestion dynamique des collisions, de la gravité, des forces viscoélastiques, etc.

L'interface haptique la plus répandue est le PHANToM®de Sensable™[MS94] (voir figure 1.3) ; elle se présente sous la forme d'un stylo dont une extrémité est montée sur des servo-moteurs. Lorsque l'utilisateur bouge le stylo, le système

[8] dispositif d'interaction qui stimule le sens du toucher et/ou applique un retour d'effort aux mouvements de l'utilisateur.

[9] du grec *haptein*, toucher

Fig. 1.3: Interface haptique PHANToM®de Sensable^{TM}.

haptique lui imprime un certain effort dépendant de la scène et des objets virtuels manipulés. On trouve également des dispositifs d'interaction sous forme d'exosquelette (voir figure 1.4). Il s'agit d'un squelette (généralement métallique) externe au corps et accroché à celui-ci par des sangles. Un ensemble de servo-moteurs ou de pistons pneumatiques contrôlent l'exosquelette et du même coup les mouvements de l'utilisateur.

Fig. 1.4: Gant exosquelette CyberGrasp^{TM}de Virtual Technologies, Inc.

On parle également d'interface haptique passive, lorsqu'il n'y aucun retour d'effort dynamique calculé par l'ordinateur. Par exemple, lorsque l'utilisateur travaille sur une donnée affichée sur une plaque de plexiglas transparente, il peut sentir son contact et sa forme.

On peut associer au toucher deux autres composantes : la proprioception et la kinesthésie.

La proprioception La proprioception est la capacité de sentir la position, l'orientation et le mouvement des parties de son propre corps. Les informations proprioceptives sont données par les récepteurs de l'oreille, qui fournissent des données d'équilibre, d'accélération, de position, d'orientation et de mouvement. Pour Slater *et al.* [SU94], la proprioception est le résultat de la formation d'un

modèle mental inconscient qu'a une personne de son propre corps et de ses mouvements. L'utilisateur a la sensation inconsciente de son corps et de ses actions dans l'espace. À l'inverse, l'**extéroception** [Han97] est la perception des phénomènes externes impliquant les sens de l'ouïe, de la vision, du toucher, etc.

La kinesthésie La kinesthésie est une modalité de la sensibilité proprioceptive. Elle est la perception consciente de la position et des mouvements des différentes parties du corps. Ce sont des informations d'origine musculaire, tendineuse, articulaire et cutanée.

La proprioception et la kinesthésie ne constituent pas un sens unitaire comme la vision par exemple. Le cortex utiliserait plutôt une combinaison de signaux émis par l'ensemble des récepteurs.

1.3.3 L'ouïe

Avec 20% de monopolisation de l'attention d'un utilisateur, l'ouïe se place juste derrière la vision. Les sons, créés par une perturbation de l'air, voyagent de la source d'émission jusqu'à l'oreille, où les vibrations sont transformées en signaux électriques et acheminés jusqu'au cerveau.

L'intensité des vibrations met en mouvement 3 osselets dans l'oreille moyenne : l'étrier, l'enclume et le marteau. Ainsi, l'oreille n'agit pas qu'en appareil d'écoute. En effet, l'étrier transmet les vibrations à travers la fenêtre ovale jusqu'à l'oreille interne. Cette dernière contient deux liquides, l'endolymphe et le périlymphe. L'endolymphe contribue au contrôle de l'équilibre. Il est ainsi possible de modifier, ou du moins perturber, l'équilibre d'un utilisateur en lui faisant parvenir des sons bien particuliers qui agissent directement sur l'oreille interne.

L'ouïe permet de se situer dans un environnement, puisqu'en repérant les sources sonores, notre cerveau est alors capable de consolider le modèle cognitif d'un lieu, à la condition que les sens fournissent des indices environnementaux qui concordent. À l'inverse, si l'ouïe permet de situer une source sonore qui ne correspond pas à ce que l'on voit, nous sommes désorientés.

Les vibrations sonores peuvent être produites artificiellement à l'aide de haut-parleurs. Et le degré de réalisme sonore dépend d'un certain nombre de facteurs : le nombre de sources, leur emplacement, etc. Lors d'une séance de cinéma par exemple, il peut y avoir jusqu'à 6 canaux avec le *Dolby Digital EX*™ : 3 voies avants, deux voies arrières stéréo, et 1 voie pour les graves. Ce système peut donner l'impression qu'une foule se presse autour du spectateur, conformément à ce qu'il voit sur l'écran. Bien entendu, il ne s'agit là que de l'exemple le plus connu, et il existe de nombreux autres techniques spécifiques au monde de la réalité virtuelle. De nombreuses recherches sont menées en acoustique sur la perception spatiale du son chez l'homme, c'est-à-dire à la simulation des indices de localisation spatiale de l'audition humaine pour des sources sonores virtuelles.

1.3.4 L'odorat

L'odorat est le quatrième des cinq sens a être étudié en réalité virtuelle. Le fonctionnement de l'odorat fut longtemps un grand mystère pour les chercheurs et ceux-ci se sont d'abord tournés vers l'étude des molécules odorantes. La distance qu'une molécule peut parcourir dans l'air dépend de son poids. Ainsi plus elles sont légères et plus elles peuvent voyager vite, jusqu'à arriver au sommet de la cavité nasale. Là, une myriade de poils filtre les molécules qui sont ensuite capturées par des cellules réceptrices. Chaque cellule nerveuse, sensible à certaines molécules, envoie un message électrique au cerveau.

Un être humain est capable de cerner environ 400 000 odeurs différentes. Il est difficile de recréer un environnement odorant virtuel, car il faut être en mesure de reproduire des odeurs particulières, à défaut de pouvoir stimuler directement les cellules nerveuses de la cavité nasale. Un certain nombre d'appareils permet de libérer des odeurs de synthèse en vaporisant dans l'air une petite quantité de molécules odorantes. Par exemple, lorsqu'une personne passera à côté d'une roseraie virtuelle, elle pourra sentir l'odeur des roses. Actuellement, plusieurs technologies de stockage des odeurs existent : liquides, gels, cires. La libération des odeurs repose essentiellement sur la vaporisation vers le nez de l'utilisateur qu'il faut ensuite nettoyer pour libérer une nouvelle odeur.

John Cater et ses collègues du Deep Immersion Virtual Environment Laboratory de l'Institut de Recherche Sud de San Antonio, sont les premiers à proposer une interface odorante spécifiquement dédiée à la réalité virtuelle. Le système se nomme le DIVEpak, et consiste en un sac à dos ; son fonctionnement repose sur la libération d'odeurs différentes dans un masque que porte l'utilisateur ; les odeurs sont libérées avec un temps de réponse de l'ordre du quart de seconde.

1.3.5 Le goût

Le goût, comme l'odorat, est un sens assez complexe à tromper. L'organe qui nous permet de goûter est la langue. Elle est tapissée de nombreuses cellules gustatives, appelées papilles. Il existe trois types de papilles : les papilles caliciformes, les foliées et les fongiformes. Sur ces papilles se trouvent des bourgeons qui permettent de discerner quatre saveurs fondamentales : le sucré, le salé, l'acide et l'amer. Le dispositif de la langue est complété par la bouche et la gorge qui contiennent également des cellules sensorielles qui participent à l'élaboration du goût.

Les cellules du goût sont sensibles à un signal chimique particulier, comme pour l'odorat. D'ailleurs les autres sens participent indirectement à l'élaboration du goût dans le cerveau, car c'est lui qui traduit les influx nerveux issus des papilles en sensation gustative. En particulier, le goût commence par se former lorsque l'aliment est approché de la bouche. Le nez sent l'odeur de l'aliment, puis lorsque celui-ci est posé dans la cavité buccale, le goût est complété par les informations des cellules dédiées au goût. Enfin, au moment d'avaler, l'odeur de l'aliment pénètre une dernière fois dans le nez, *via* l'arrière de la cavité nasale. Le cycle de formation du goût est alors terminé.

Des simulateurs d'aliments existent, mais ils restent pour le moment anecdotiques. Les aliments sont d'abord analysés sous toutes les coutures (force nécessaire pour mordre l'aliment, constituants chimiques, ...). Puis, une interface est placée dans la bouche et libère une mixture de molécules sur la langue. En même temps, des molécules odorantes sont vaporisées en direction du nez de l'utilisateur. Hiroo Iwata de l'Université de Tsukuba au Japon et ses collègues proposent le système Food Simulator [IYUM04]. Ce simulateur de goût se place dans la cavité buccale de l'utilisateur. Il libère une odeur, un goût, et émet des bruits de mastication qui correspondent à l'aliment prétendument mastiqué.

1.4 Conclusion

De prime abord, la réalité virtuelle semble être une technologie de pointe assez récente. Mais en fouillant un peu, force est de constater que ce domaine prend ses racines bien avant l'ère de l'ordinateur. En fait, l'homme s'est aperçu qu'il pouvait altérer et déformer la réalité, en superposant au regard de l'être humain, une autre réalité, celle-ci fabriquée de toute pièce. À ce titre, on pourrait déjà voir l'invention des images animées, puis du cinéma, comme autant de moyens mis en œuvre pour réussir le pari audacieux de voir les images d'un monde qui n'existe pas vraiment. Et c'est bien cela la réalité virtuelle : une stimulation sensori-motrice de l'être humain permettant à l'esprit de croire en l'existence d'un univers qui n'a pas cours matériellement.

Nous venons de le voir dans ce chapitre, les sens de l'homme construisent véritablement le réel. En premier lieu la vision qui a disposé d'un nombre incroyable de recherches pour améliorer les dispositifs de création d'images. Aujourd'hui, certains laboratoires commencent à toucher du doigt des dispositifs d'affichage holographiques, produisant des images véritablement volumiques. Pourtant, en réalité virtuelle, les autres sens demeurent relativement sous-exploités, et de très gros efforts devront être réalisés avant de disposer d'interfaces matérielles suffisamment performantes pour stimuler efficacement toutes les sensibilités sensorielles de l'être humain.

Pour que l'interaction entre l'homme et la machine soit performante et efficace, il faut réduire au maximum le nombre d'intermédiaires entre les deux. Il est essentiel de bien comprendre les mécanismes sensoriels de l'être humain, et en particulier, comment un modèle cognitif peut prendre forme à partir des informations du monde extérieur. On sera alors en mesure de savoir comment stimuler les sens en disposant d'interfaces matérielles miniaturisées et autonomes, jusqu'à ce que l'utilisateur ne puisse plus se rendre compte de leur présence.

La réalité virtuelle est à la croisée de bon nombre de domaines de recherche. En premier lieu l'informatique, mais aussi la robotique, la physique, les neurosciences, et encore bien d'autres spécialités. Cette activité multi-disciplinaire doit être prise comme telle, et ne pas négliger de s'intéresser à tous les aspects qu'elle permet d'entrevoir.

Dans les deux chapitres qui suivent, nous avons pris le parti de considérer à

la fois les aspects matériel et logiciel. En effet, il nous semble important d'appréhender l'interaction en réalité virtuelle comme un domaine à deux composantes interdépendantes. Le défi qui se pose alors aux chercheurs est de proposer tous les mécanismes matériels qui permettent à l'utilisateur de sentir et modifier le monde réel, tout en insérant les aspects logiciels entre le matériel et l'homme.

Chapitre 2

Interaction $3D$

2.1 Introduction

Dans ce chapitre nous nous intéressons aux différentes formes d'interaction que l'on trouve en réalité virtuelle, pour les tâches les plus courantes. L'interaction $3D$ est le liant entre les périphériques d'interaction, qui fournissent des informations telles que leur position ou leur orientation, et les logiciels qui traitent ces données en leur ajoutant un comportement particulier. Plusieurs auteurs se sont intéressés à classifier toutes les méthodes d'interaction existantes pour en dégager une taxonomie générale. Nous commençons par présenter, section 2.2, les grandes classifications que l'on retrouve dans la littérature, ainsi qu'une description succincte des tâches que nous avons retenues.

Après avoir présenté notre classification générale, nous détaillons chacune des tâches dans une section. Nous voyons la plupart des techniques qui existent[1], en indiquant le fonctionnement de chaque méthode, et pour certaines, nous fournirons quelques détails importants sur l'implantation. Notre choix de mêler aux outils d'interaction logiciels certains dispositifs physiques pourra dérouter le lecteur. Cependant, il nous semble important de discuter de certains périphériques physiques, tant la séparation entre le matériel et le logiciel est parfois ténue.

La section 2.3 aborde la navigation dans un environnement virtuel. Nous montrons que cette tâche consiste à la fois à *trouver son chemin* – s'orienter, et à *voyager* – se déplacer. Nous nous attardons essentiellement sur les méthodes de déplacement, puisque ce sont celles qui nous intéressent le plus ; sans omettre toutefois certaines méthodes dédiées à l'orientation dans une scène virtuelle lorsqu'une technique de déplacement le nécessite.

Après que l'utilisateur se soit correctement positionné dans une scène virtuelle, il souhaite généralement manipuler un ou plusieurs objets. La section 2.4 traite de la tâche de sélection, préalable nécessaire à la manipulation. La sélection est décomposable en deux sous-tâches : la *désignation* du ou des objets d'intérêts, puis la *validation de cette désignation*. Bien que de nombreux auteurs choisissent de regrouper sélection et manipulation en une tâche commune, nous les avons séparées volontairement, bien que certaines techniques d'interaction

[1] Nous ne saurions être exhaustifs sur ce sujet. Néanmoins nous espérons couvrir le maximum de techniques d'interaction rencontrées en réalité virtuelle.

ne fassent pas la distinction. Il y a donc quelques redondances dans la section suivante consacrée à la manipulation.

La manipulation fait l'objet de la section 2.5. Très liée à la sélection, cette tâche consiste en la modification directe de certains paramètres de l'objet préalablement sélectionné. Par exemple, le déplacement et l'orientation, ou encore la mise à l'échelle d'un objet sont des tâches directes que l'on réalise très souvent dans la vie courante. Nous voyons que de nombreux outils logiciels permettent à l'utilisateur de manipuler tout aussi bien des objets distants que ceux posés à portée de main.

Lorsque l'on souhaite manipuler certains objets qui modifient un ou plusieurs états de l'application, et plus généralement du système de réalité virtuelle, on utilise des outils dédiés au contrôle du système. Anciennement appelé contrôle d'application, le contrôle du système prend place dans la section 2.6. Le plus souvent, cette tâche repose sur des objets dont l'ensemble forme ce que l'on nomme couramment l'interface utilisateur[2], comme par exemple les boutons et les menus.

La dernière tâche d'interaction concerne l'édition et l'entrée de symboles, c'est-à-dire de lettres, de chiffres, et de n'importe quel autre symbole ; elle sera développée section 2.7. Cette dernière tâche d'interaction est assez récente, et bien qu'elle soit activement utilisée dans les environnements de bureau, elle reste pour le moment peu étudiée en réalité virtuelle.

2.2 Les différentes formes d'interaction en réalité virtuelle

Entre les années 1990 et 2000, la recherche en matière d'interaction fut particulièrement prolifique, bien qu'elle se soit sensiblement reduite depuis. En effet, les chercheurs se concentrent désormais sur des outils spécifiquement dédiés à une application particulière. Durant cette période, les chercheurs se concentraient surtout à imaginer des techniques d'interaction innovantes, sans forcément les rattacher à une application spécifique. De cette débauche d'imagination, le nombre de techniques est suffisamment important pour les regrouper par genre en les classant selon certains critères généraux.

Il existe plusieurs classifications, et chacune se situe à un niveau de généralité différent. Certains auteurs situe l'interaction au niveau comportemental, comme par exemple dans l'ouvrage de Fuchs *et al.* [FMBC06]. Nous souhaitons placer notre classification à un niveau plus technique, en considérant plus les actions de l'utilisateur sur une scène virtuelle que son comportement ou son intention.

C'est Mine [Min95] qui propose le premier une classification basée sur quatre formes fondamentales d'interaction dans un environnement virtuel : la navigation, la sélection, la manipulation et la mise à l'échelle. Notons que la mise à l'échelle est séparée de la manipulation, ce qui peut surprendre aujourd'hui. Il

[2]a. *GUI* pour *Graphical User Interface* lorsque l'interface est représentée visuellement.

a également défini une cinquième forme qui dérive des quatre précédentes : l'interaction par widgets et menus virtuels, que l'on nomme maintenant contrôle du système. Par la suite, [Han97] introduit ce qui constituera la base de la classification moderne, reprise et étendue plus tard par Bowman [Bow99] dans son mémoire de thèse, chaque technique trouvant sa place dans une des 4 grandes tâches : la navigation, la sélection, la manipulation et le contrôle du système. Il s'agit des quatre formes d'interaction couramment utilisées en réalité virtuelle. Nous pouvons voir que les trois premières sont proches de ce dont nous avons l'habitude dans notre réalité quotidienne. Nous retiendrons donc ces quatre formes d'interaction auxquelles nous rajoutons la tâche d'*entrée de symboles*. Cette dernière regroupe toutes les techniques qui permettent de saisir des symboles comme des chiffres et des lettres. C'est un domaine encore peu étudié, mais qui est néanmoins essentiel pour l'interaction.

Cette classification correspond bien à nos travaux. Nous allons maintenant introduire les 5 tâches que nous avons retenues, avant d'en donner le détail, pour chacune d'elle, dans les sections qui suivront.

2.2.1 La navigation

La navigation est un processus complexe, peu évident à mettre en œuvre en réalité virtuelle et fortement dépendant des périphériques d'interaction qui sont utilisés. Pour Rheingold [Rhe91], la navigation se définit comme *la capacité à bouger à l'intérieur d'une scène générée par ordinateur*. Pour cet auteur, le terme naviguer est assez imprécis et est employé pour décrire un large panel de techniques de déplacement différentes.

Mackinlay *et al.* [MCR90] distinguent deux types de mouvements de la part de l'utilisateur. D'une part, il y a le *mouvement général*, où l'utilisateur contrôle la position du périphérique de vision en n'importe quel point de l'espace, et en ne suivant aucun chemin calculé. D'autre part, il y a le *mouvement contrôlé*, par exemple, celui qui permet d'atteindre une cible, de suivre une trajectoire, etc.

Pour Herndon *et al.* [HvDG94], la navigation est *la planification et l'exécution d'un voyage à travers l'espace, réel ou virtuel, et réalisé à l'aide de références mentales de points de repères enregistrés dans l'espace traversé*. La navigation en environnement virtuel est assez complexe. Il faut en effet résoudre le problème du repérage et celui du contrôle de la navigation (vitesse, accélération, ...). Pour ces raisons, les auteurs préconisent de rajouter des repères visuels qui complètent la compréhension (c'est-à-dire son modèle mental) du monde virtuel. Malheureusement, les informations visuelles nécessaires à une bonne orientation et à un bon équilibre du navigateur ne sont pas encore très bien connues.

L'aspect cognitif de la navigation est abordé par Rodolph P. Darken *et al.* [DAA98, DAA99]. Pour eux, la représentation mentale de l'espace constitue une carte cognitive[3], c'est-à-dire une vue de l'environnement schématisée par l'esprit de l'utilisateur. Les auteurs mettent en évidence l'importance de la vision périphérique, car elle fournit des éléments visuels (des repères visuels) qui rendent

[3] a. *cognitive map*

les déplacements plus faciles. À l'inverse, lorsque l'utilisateur ne dispose plus de tels repères pour s'orienter dans l'espace, il se retrouve complètement perdu. Ces références visuelles semblent donc apporter un réel avantage lors du processus de navigation.

Certains auteurs, comme Marsch *et al.* [MS01] s'intéressent également aux problèmes de désorientation des utilisateurs dans une scène virtuelle. Ils supposent que la désorientation repose sur deux problèmes : d'une part le manque de points de référence dans la scène (repères), et d'autre part la navigation qui a souvent lieu trop près des objets virtuels de la scène. Ils constatent que la présence de lignes guides et la détection des collisions avec les objets permettent d'atténuer ces problèmes. Les auteurs préconisent de s'assurer de la présence de deux types de lignes guides :

- des points d'entrée et de sortie qui permettent de se déplacer librement dans la scène tout en étant implicitement assisté : portes, chemins, routes, etc. ;
- le placement des objets virtuels devrait être tel qu'il y ait à tout instant au moins un objet connu, partiellement visible, dans la scène.

Ces lignes guides aident à construire la carte cognitive que l'utilisateur a de l'environnement virtuel, en étant fondues dans le décor. En particulier, elles permettent d'anticiper ce que l'on ne voit pas encore.

Lorsque l'on considère la navigation comme étant le déplacement du point de vue de l'utilisateur, comme le font Faisstnauer *et al.* [FSS97, FSS98], on peut distinguer deux modes de navigation. D'une part, le mode *à la première personne*, lorsque la caméra est à la place des yeux de l'utilisateur. Celui-ci bouge directement sa tête pour naviguer. Et d'autre part, le mode *à la troisième personne*, quand l'utilisateur manipule la caméra, c'est-à-dire son propre point de vue, à l'aide d'un avatar.

Bowman *et al.* [BKH97] considèrent que la navigation est constituée de deux caractéristiques plus ou moins distinctes :

- **le déplacement** : c'est le contrôle du mouvement du point de vue de l'utilisateur dans l'environnement ;
- et **l'orientation, la découverte d'un chemin** : il s'agit de la détermination d'un chemin à partir d'éléments visuels, de la connaissance de l'environnement.

Pour ces auteurs, plusieurs facteurs influent sur la qualité d'une technique de navigation :

- la vitesse d'utilisation ;
- la précision ;
- la connaissance implicite de sa position et de son orientation dans l'environnement pendant et après la navigation ;
- la facilité d'apprentissage ;
- la facilité d'utilisation ;
- la capacité de l'utilisateur à obtenir activement des informations de l'environnement pendant le voyage ;
- la sensation de présence ou le degré d'immersion.

Ils donnent une taxonomie des techniques de navigation en environnement immersif dans [BH99, BKLJP01].

Trouver son chemin est la capacité à s'orienter vers un endroit particulier d'une façon avantageuse et à identifier la destination quand elle est atteinte [ENK97]. Les auteurs relèvent que les points de repères aident à structurer un environnement et fournissent des indices directionnels pour faciliter la découverte du chemin. Soit la recherche est naïve (découverte) c'est-à-dire que l'utilisateur n'a aucune connaissance à l'avance de l'environnement. Soit la recherche peut être amorcée : le voyageur connaît la destination et peut bouger directement en utilisant ses connaissances et les repères sur le terrain.

Dumas *et al.* [DPC99] définissent la navigation comme *l'ensemble des déplacements de l'utilisateur à l'intérieur de l'espace* $3D$, ce qui se traduit par des changements de point de vue sur la scène. Les auteurs donnent un certain nombre de recommandations à suivre pour que l'utilisateur puisse interagir au mieux avec son environnement. Il doit d'abord travailler dans un espace aux dimensions limitées et avoir un point de vue fixe sur la scène globale avant de manipuler pour éviter des problèmes de désorientation et réorientation. Ensuite, l'interface d'interaction doit être simplifiée au maximum. Enfin, il est important de renforcer les détails visuels qui permettent à l'utilisateur de s'orienter correctement dans l'espace : placer un quadrillage au sol pour percevoir la profondeur, utiliser les ombres portées sous les objets virtuels. Ces dernières constituent les indices visuels les plus indispensables.

Laura Arns [Arn02] a, quant à elle, retenu trois composantes majeures dans la navigation, plutôt que seulement se repérer et naviguer :
- l'intégration du chemin : capacité d'un humain à mentalement mettre à jour sa position et son orientation en s'appuyant sur l'évolution de sa propre accélération et vélocité dans le temps ;
- le pilotage : ce terme se réfère à la capacité de mettre à jour sa propre position et orientation par rapport à d'autres positions. Par exemple, être capable de se placer sur une carte. Le pilotage est ici synonyme de trouver son chemin ;
- la locomotion : quels sont les moyens que l'on utilise pour se mouvoir dans une scène virtuelle.

Définition La navigation regroupe l'ensemble des paradigmes d'interaction qui autorisent l'utilisateur à se mouvoir dans un environnement virtuel.

Bien que cette tâche contienne également la possibilité de trouver son chemin, nous n'aborderons que très peu cet apsect dans la section 2.3, consacrée aux métaphores de navigation.

2.2.2 La sélection

Bowman *et al.* [BDHN99, BH99] définissent la sélection comme la désignation d'un ou plusieurs objets virtuels pour :

- déclencher une commande ;
- définir un argument ;
- modifier une propriété ;

– manipuler un objet.

La sélection n'est pas un terme bien délimité. Ainsi, il arrive couramment que l'on regroupe sous le terme sélection la tâche de désignation proprement dite, et la tâche de validation de la désignation. Par exemple, dans un environnement fenêtré, quand l'utilisateur passe sa souris sur une icône, il la désigne. Mais, ce n'est que lorsqu'il clique sur le bouton de la souris, et que le pointeur se trouve sur l'icône, que la sélection devient effective. Cette tâche est le préalable nécessaire à la manipulation, et certains auteurs regroupent d'ailleurs les deux.

Définition La sélection est la tâche d'interaction qui permet à la fois de désigner un ou plusieurs objets dans une scène virtuelle, puis de valider cette désignation.

2.2.3 La manipulation

La manipulation est le processus par lequel l'utilisateur réalise une transformation de l'objet qu'il a sélectionné (rotation, translation, mise à l'échelle, ...). De manière plus générale, la manipulation peut être vue comme étant le processus d'édition/modification des propriétés d'un objet. Dans la littérature, sélection et manipulation sont souvent regroupées en une seule et même tâche.

Précisons que la tâche de manipulation peut être directe ou indirecte. La manipulation directe est celle où l'utilisateur opère directement sur l'objet affiché à l'écran. La manipulation indirecte est celle où l'utilisateur modifie les paramètres de l'objet par l'intermédiaire d'entités graphiques (boutons, menus, ...), matérielles, etc.

Définition La manipulation regroupe l'ensemble des paradigmes d'interaction qui permettent de modifier un ou plusieurs attributs d'un objet virtuel, comme par exemple sa position, son orientation ou encore sa taille.

2.2.4 Le contrôle du système

Le contrôle du système englobe toutes les modifications qui interviennent sur l'état et/ou le mode de l'application, ou de façon plus générale, du système, souvent réalisées à l'aide de menus ou de commandes en ligne. Bowman *et al.* [BW01] définissent le contrôle du système comme étant les *changements de l'état du système ou du mode d'interaction*. Pour l'heure, la technique d'interaction la plus étudiée demeure les menus, qui sont extrêmement populaires dans les interfaces graphiques $2D$.

Cependant, on peut se poser la question de la pertinence d'utiliser des menus dans les environnements virtuels : en définitive, sont-ils appropriés pour contrôler le système ? A-t-on besoin de tous les degrés de liberté offerts par une interaction à six degrés de liberté quelle que soit la tâche à réaliser ? Le contrôle du système a pour l'instant été assez peu étudié par rapport aux tâches de navigation, de sélection et de manipulation.

Définition L'application, l'environnement et les données qui y sont rattachées forment un ensemble cohérent que l'on nomme système. Le contrôle du système rassemble les métaphores d'interaction qui agissent directement ou indirectement sur l'entité « système ».

2.2.5 L'entrée de symboles

Le contrôle du système, anciennement contrôle d'application, a été récemment scindé en deux parties par la communauté virtuelle : d'une part **le contrôle du système** (voir paragraphe précédent) et d'autre part, **l'entrée de symboles**. Cette dernière tâche d'interaction correspond à l'édition de texte, de nombres et d'autres types de symboles. Alors que sur un environnement du bureau, le clavier est tout indiqué, il est assez délicat d'utiliser ce dernier dans un environnement immersif, du fait de l'encombrement et de la gène qu'il engendre. Il s'agit d'un domaine assez peu étudié pour le moment, par rapport à la navigation, la sélection et la manipulation.

Définition L'entrée de symboles regroupe l'ensemble des paradigmes qui permettent d'entrer et d'éditer des chiffres, des caractères et tout autre type de symboles.

2.3 La navigation

Dans un environnement, quelle que soit sa nature, l'interaction passe avant tout par la navigation, c'est-à-dire tout ce qui concerne le déplacement de l'être humain. Pour [BKH97], la navigation se compose de deux tâches fondamentales :

- le déplacement : c'est le contrôle du mouvement du point de vue de l'utilisateur dans l'environnement ;
- l'orientation, la découverte d'un chemin : il s'agit de la détermination d'un chemin à partir d'éléments visuels, et de la connaissance de l'environnement.

Nous allons voir un ensemble de métaphores de navigation utilisées en réalité virtuelle. Nous présentons les techniques de déplacement dans une scène virtuelle, et non celles qui servent à s'orienter. Sous cette restriction, la navigation va de tourner sa tête pour observer un objet sous un autre angle, à déplacer tout son corps dans un environnement, quelle que soit l'échelle de ce dernier. Nous avons choisi de scinder les métaphores de déplacement en 5 familles : celles qui se font par mouvements physiques (section 2.3.1), celles qui contrôlent le point de vue (section 2.3.2), celles qui indiquent la direction (section 2.3.3), celles qui permettent à l'utilisateur de s'accrocher à quelque chose pour se déplacer (section 2.3.4), et ,enfin, celles qui permettent de se déplacer selon un chemin planifié (section 2.3.5).

2.3.1 Navigation par mouvements physiques

Définition La navigation par mouvements physiques désigne les techniques de déplacements basées sur l'utilisation du corps de l'utilisateur. L'emploi des membres inférieurs est prépondérante dans ce type de navigation, car ils offrent

un parallèle direct avec les déplacements dans le monde réel.

Les mouvements physiques apparaissent comme le choix le plus logique pour se déplacer, car nous les utilisons tous les jours pour nous mouvoir dans le monde réel. Nous allons voir que les différentes formes de déplacements physiques reposent essentiellement sur l'utilisation des jambes et des bras. La marche réelle peut être employée lorsque le lieu est suffisamment vaste et sécurisé. Quand ce n'est pas possible, la marche sur place, avec ou sans tapis propose un palliatif intéressant. D'autres solutions plus exotiques, comme le suivi des mouvements de la tête ou l'utilisation de la bicyclette, sont également proposées. Nous conseillons la lecture de Hollerbach [Hol02] qui détaille les différents périphériques d'interaction (tapis, vélos, chaussons, ...) mis en œuvre pour de tels déplacements.

Marche réelle

La technique la plus simple à mettre en œuvre pour se déplacer dans un univers $3D$ est la marche physique, puisqu'elle est naturelle, et permet au marcheur de bien comprendre son environnement – notamment à cause des sensations de déplacement perçues par son système vestibulaire.

Ware *et al.* [WO90] ont présenté une des premières métaphores de navigation en réalité virtuelle, basée sur la marche réelle. Ils définissent le modèle du Cyberespace dans lequel l'utilisateur change son point de vue en marchant physiquement et en tournant sa tête. Cette métaphore a la particularité d'être simple et naturelle. En outre, elle offre une bonne sensation de présence dans la scène virtuelle. Cependant, elle souffre d'un certain nombre de limitations. Tout d'abord, les déplacements de l'utilisateur sont restreints à la pièce où il se trouve, à la distance maximale de déplacement offerte par les périphériques (lunettes, gants de données, ...), et à la nature et l'encombrement du sol. De plus, la marche réelle permet de ne faire que des mouvements naturels, alors qu'il serait intéressant de disposer de mouvements comme le vol et la téléportation. Enfin, lorsque le matériel de suivi des mouvements et les périphériques d'interaction sont reliés à l'aide de câbles, cela limite les possibilités de déplacement.

Razzaque *et al.* [RKW01] ont imaginé une technique astucieuse permettant de s'affranchir du problème des déplacements réels limités : la marche redirigée[4]. L'utilisateur se trouve placé au centre d'une pièce aux dimensions réduites. La scène virtuelle affichée devant ses yeux est tournée imperceptiblement et en permanence. Ainsi, l'utilisateur est obligé de corriger en permanence sa trajectoire réelle, sans s'en rendre compte, de manière à ce qu'il se dirige toujours vers le mur physique qui se situe le plus loin de lui (voir figure 2.1). Il peut ainsi parcourir de grandes distances en utilisant la marche réelle, et ce sans souffrir de la taille de la pièce. Cette technique n'est envisageable que pour les environnements immersifs de type CAVE à 3 murs minimum. Il est par conséquent impossible de l'utiliser sur un plan de travail de type *Workbench*.

Une autre solution a été donnée par Welch *et al.* [WBV+99]. Leur système

[4]a. *Redirected Walking*

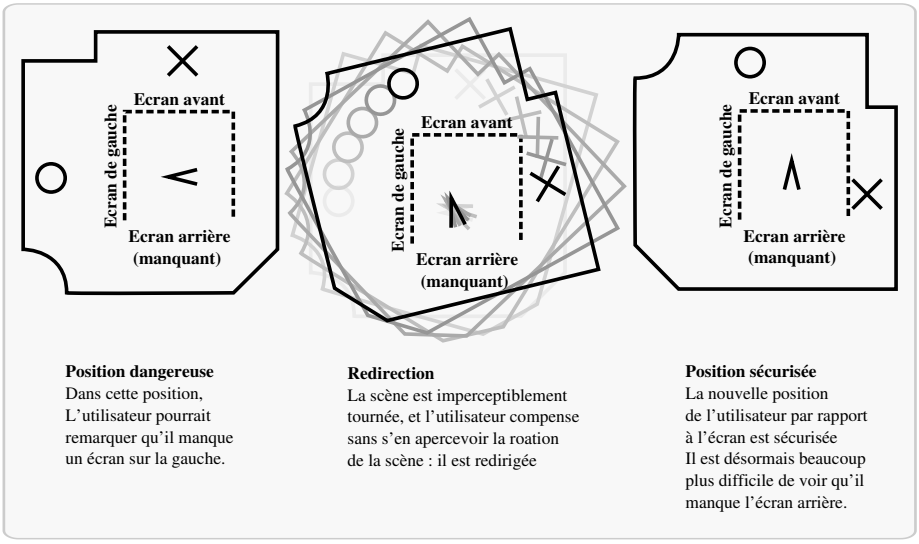


Fig. 2.1: Principe de la marche redirigée : en traits pleins, la scène virtuelle, et en pointillés, le CAVE^{TM}.

HiBall permet de suivre les mouvements de la tête de l'utilisateur dans des pièces de 5 par 10 mètres de côté, voire encore plus grandes. Il est constitué d'un ensemble de caméras miniatures placées sur la tête de l'utilisateur (c'est le HiBall), ainsi que de diodes infrarouges fixées au plafond de la pièce. Ces dernières émettent des flashs à intervalles réguliers, qui sont ensuite lus par les caméras du HiBall, ce qui permet de calculer l'orientation et la position de la tête de l'utilisateur dans l'espace. Les utilisateurs n'ont plus de problèmes de câbles qui les gênent, et la taille maximale de la pièce n'est limitée que par le nombre de diodes placées sur le plafond.

L'usage du suivi du mouvement des membres inférieurs du corps est curieusement peu étudié en réalité virtuelle. LaViola *et al.* [LJAFKZ01] utilisent les Interaction Slippers, des chaussons munis de traqueurs que l'utilisateur enfile à ses pieds. Les auteurs estiment qu'il vaut mieux utiliser les pieds que les mains pour se déplacer dans un environnement virtuel. En effet, les mains sont libres pour réaliser une autre tâche et les pieds sont les membres dédiés au déplacement dans le monde réel. La métaphore de navigation associée se nomme Step World In Miniature, en référence au World In Miniature de Stoakley *et al.* [SCP95]. Le Step WIM est une carte miniature de l'ensemble de la scène qui est affichée sur le sol. Pour se déplacer, l'utilisateur positionne ses chaussons à un endroit précis de la carte. Il est également possible de simuler la pression d'un bouton lorsque les deux chaussons se touchent. Par exemple, pour se déplacer, l'utilisateur appuie un pied contre l'autre, ce qui fait apparaître le Step WIM. Puis, il se déplace, et réitère un contact entre les deux chaussons pour faire disparaître la métaphore. Bien entendu, lorsque la carte est affichée, l'utilisateur doit veiller à ne pas bouger les pieds lorsque ceux-ci touchent le sol, sous peine de se déplacer sans le vouloir.

La marche réelle est une technique simple à mettre en œuvre et intuitive à utiliser. Cependant, dans la plupart des applications de réalité virtuelle, elle n'a pas sa place, dans la mesure où les environnements immersifs sont de taille réduite. À l'inverse, en réalité augmentée, les espaces où l'utilisateur peut se déplacer sont généralement plus vastes. La marche réelle est plus adaptée à ce cas là.

Marche sur place

Certains auteurs ont contourné le problème de limitation des déplacements physiques. Slater *et al.* [SU94] ont imaginé qu'il suffisait de marcher sur place plus ou moins vite, sans déplacement du corps. La direction du déplacement est donnée par le torse. Cette technique de déplacement semble être un bon compromis entre la marche sur place et les autres techniques dites magiques – comme le vol ou la téléportation – que l'on rencontre couramment dans les environnements virtuels. La sensation de présence est bien conservée, car il y a toujours utilisation de la marche physique. D'autre part, il n'y a plus de limitation relative à la taille de la pièce. Les auteurs ont également imaginé une variante pour grimper virtuellement sur des obstacles : l'utilisateur marche sur place et il lève sa main pour faire le même mouvement que s'il souhaitait monter une échelle.

Les auteurs améliorent cette technique de navigation en utilisant un réseau de neurones [SUS95] qui interprète le mouvement vertical des pieds lors de la marche sur place pour déterminer si l'utilisateur se déplace ou non, ce qui évite de détecter tous les mouvements parasites. La direction est donnée par le regard et non plus par le torse. La métaphore de l'échelle repose également sur l'utilisation d'un réseau de neurones pour analyser les mouvements des mains de l'utilisateur pour déterminer s'il monte ou s'il descend. Des études menées par Usoh *et al.* [UAW+99] ont montré que la marche sur place accroît la sensation de présence par rapport à un déplacement purement virtuel, comme le vol et la téléportation, mais de manière moins importante que la marche réelle.

Le système GAITER (voir figure 2.2) de Templeman *et al.* [TDS99], est similaire à ce que LaViola *et al.* [LJAFKZ01] ont proposé : cette fois, les chaussures de l'utilisateur sont munies de capteurs qui permettent de détecter s'il y a une pression qui s'exerce, c'est-à-dire s'il y a un contact entre le pied et le sol. Il est ainsi possible de suivre les mouvements des pieds et de mettre en place un système de marche sur place. Cette technique permet d'avancer, de reculer et de tourner sur place. La cadence des pas virtuels permet d'accélérer ou de ralentir les déplacements.

Bien qu'elle augmente la sensation de présence, la technique de la marche sur place souffre de problèmes de reconnaissance des mouvements des pieds et de la gêne occasionnée par les câbles. Elle convient bien aux environnements où l'espace est réduit, et lorsqu'on souhaite disposer d'une technique naturelle et réaliste.

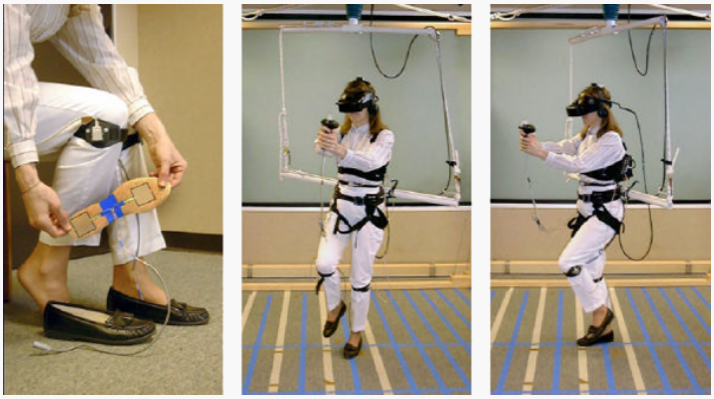
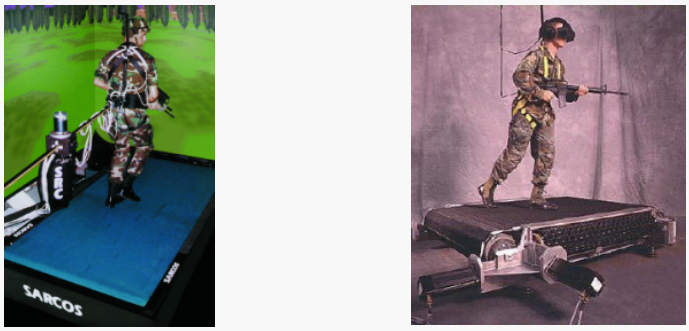


Fig. 2.2: Système GAITER de Templeman Naval Research Labs.

Tapis de marche

Il est possible de pallier le problème de la taille limitée de l'espace de déplacement en ayant recours à des périphériques dédiés à la marche réelle en environnement limité. On trouve les techniques basées sur les tapis roulants : tapis de marche unidirectionnel (figure 2.3 a.) ou omnidirectionnel (figure 2.3 b.).



a. tapis SARCOS a. tapis IWATA

Fig. 2.3: Tapis unidirectionnel SARCOS (à gauche) et omnidirectionnel IWATA (à droite).

Avec les tapis de marche, l'utilisateur se déplace – éventuellement dans plusieurs directions si le tapis est omnidirectionnel, alors que son corps reste sur place. Le principal inconvénient de ces dispositifs est qu'il faut pouvoir détecter le moment où l'utilisateur souhaite changer de direction. Souvent, même pour les tapis omnidirectionnels, la modification de trajectoire apparaît brutale et

manque de réalisme.

Des recherches ont été menées pour corriger ce problème. Une solution assez sophistiquée consiste à suivre les mouvements des pieds de l'utilisateur pour savoir lorsqu'il souhaite tourner. Lorsqu'un changement de direction est enregistré, le tapis roulant omnidirectionnel est pivoté de manière à ce qu'il reste dans le sens de marche de l'utilisateur. Le tapis ATR ATLAS de Noma *et al.* [NM98] est monté sur une structure qui peut pivoter dans toutes les directions, autour d'un axe vertical. Leur système permet également de simuler une pente dans le sens de la marche.

Les auteurs ont étendu leur tapis [NSM00] et proposé un dispositif de marche assez original, l'ATR GSS, qui peut se déformer pour simuler un relief plus ou moins accidenté. La surface est composée d'un ensemble de vérins hydrauliques qui se soulèvent ou s'abaissent selon la déclivité du terrain.

Les tapis omnidirectionnels autorisent généralement les déplacements transversaux, comme lorsqu'on fait un pas sur le côté. Malheureusement, lors d'un tel mouvement, l'utilisateur peut se retrouver déséquilibré. En effet, il faut noter que de tels périphériques mettent un certain temps pour détecter les mouvements, ce qui n'est pas sans risque pour l'utilisateur, puisqu'il peut chuter. De plus, il est important que le marcheur se trouve assez loin des bords du tapis, et soit recentré en permanence au milieu du système pour éviter de tomber.

Suivi des mouvements de la tête

Lorsqu'un individu marche réellement, sa tête effectue un mouvement bien spécifique sans même qu'il s'en rende compte, en se balançant d'avant en arrière. Inversement, ce mouvement peut être réalisé par l'utilisateur pour simuler une marche réelle, bien que les jambes restent immobiles. Il sera analysé par le système (par un réseau de neurones par exemple), et permettra de se déplacer simplement dans une scène virtuelle.

Song *et al.* [SN92] présentent une nouvelle métaphore de navigation basée sur le suivi des mouvements de la tête. Une fonction non linéaire de déplacement est associée au mouvement réel de la tête. Lorsque l'utilisateur bouge la tête en dessous d'une certaine distance d par rapport au début du mouvement, le déplacement est constant. Puis, lorsque la distance *début du mouvement - position actuelle de la tête* dépasse d, alors le déplacement du point de vue augmente de façon exponentielle. En faisant le parallèle avec la course à pied, plus le coureur va vite, plus sa tête bouge, et c'est ce principe qui est repris dans cette technique de déplacement.

D'autres auteurs, comme Bourdot *et al.* [BDA99] s'intéressent également au suivi des mouvements de la tête : la tête de l'utilisateur est suivie, le système analyse les objets qui semblent intéresser l'utilisateur et déplace l'avatar de ce dernier en fonction des déplacements de la tête. Les auteurs justifient leur technique principalement dans le but de libérer les mains pour d'autres tâches que la navigation.

L'inconvénient majeur des deux techniques de suivi des mouvements de la tête que nous venons de voir provient de l'interprétation que le système fait. En effet, il arrive que l'utilisateur ne souhaite pas se déplacer, mais juste observer certains détails de la scène virtuelle. Éventuellement, lorsqu'il acquiesce en discutant avec une autre personne, il peut hocher sa tête d'avant en arrière. Il est difficile de différencier ce mouvement d'une réelle volonté de naviguer.

Vélos

La marche réelle ou l'utilisation d'un tapis ne sont pas toujours possibles, notamment par manque de place. Nous avons également vu que le suivi des mouvements de la tête a certaines limitations. La bicyclette représente une alternative intéressante que certains auteurs ont choisi d'étudier. Là encore, il s'agit de techniques qui reposent sur l'emploi des jambes et qui consituent une approche assez naturelle en terme d'interaction.

Brogan *et al.* [BMH98] proposent deux simulations de déplacement à l'aide d'un vélo. Dans la première, l'utilisateur doit pousser des moutons virtuels pour les faire rentrer dans un parc clôturé. Avec leur système, la bicyclette permet de naviguer rapidement d'un endroit à un autre dans la scène virtuelle. En même temps l'utilisateur reste assis et calé sur la selle. La direction est donnée *via* les mains par l'orientation du guidon.

Dans la deuxième simulation, le système Hodgins développé à l'Université Georgia Tech (figure 2.4 a.) permet de participer à une course olympique virtuelle. L'utilisateur doit bien entendu pédaler aussi vite que possible pour dépasser ses concurrents synthétiques, et se servir de ses mains pour orienter le guidon, ce qui détermine sa trajectoire. Le dispositif permet d'orienter la bicyclette vers le haut ou vers le bas pour simuler une montée ou une descente.

L'utilisation de la bicyclette est certes intéressant puisqu'il est possible d'avancer ou reculer, plus ou moins, en pédalant en avant ou en arrière. Mais les mains sont occupées à tenir le guidon et à indiquer la direction du déplacement. L'utilisateur ne peut donc pas faire autre chose en même temps qu'il navigue.

Le système Uniport proposé par Darken *et al.* [DCC97] pour l'armée américaine est un monocycle qui peut pivoter sur un axe vertical (figure 2.4 b.). Il constitue une alternative à la nécessité d'un guidon. L'utilisateur est sanglé sur la selle et pédale en avant ou en arrière ; pour tourner il lui suffit de pencher son buste d'un côté et la scène virtuelle pivote alors en conséquence. Ce système est cependant assez fatigant à utiliser lorsque l'interaction se prolonge dans le temps.

Lorsque les mouvements physiques sont impossibles, par exemple par manque de place, ou tout simplement inadaptés à certaines applications, on peut envisager de contrôler le point de vue. Le point de vue est alors donné par une caméra que l'utilisateur manipule directement ou indirectement. C'est ce que nous allons étudier dans les paragraphes qui suivent.

a. bicyclette Hodgins

a. monocyle Uniport

Fig. 2.4: Système de navigation Hodgins à base de bicyclette de l'Université Georgia Tech (à gauche) et Uniport (à droite).

2.3.2 Navigation par contrôle du point de vue

Définition La navigation par contrôle du point de vue consiste à modifier directement ou indirectement la caméra qui voit la scène. La caméra peut être centrée sur l'utilisateur, on parle alors de vue subjective. Elle peut également constituer un second point de vue que l'utilisateur peut manipuler à loisir.

Nous nous intéressons au contrôle du point de vue selon plusieurs aspects ; celui-ci peut être manipulé directement, comme si l'on tenait une caméra en main. Nous voyons que le point de vue peut également être représenté par une scène miniature. Puis nous étudions les techniques qui changent l'échelle du monde comme si l'utilisateur se rapprochait ou s'éloignait dans la scène. Enfin, nous abordons les techniques dites de lentilles magiques qui permettent de se focaliser sur une partie bien précise du monde virtuel.

Manipulation directe du point de vue

La manipulation directe du point de vue est assez simple : en orientant sa main dans l'environnement virtuel, il est possible de se déplacer à l'aide d'un simple traqueur. Ware *et al.* [WO90] ont proposé la technique de la caméra dans la main[5]. Un traqueur est fixé à la main de l'utilisateur et donne en permanence des informations de position et d'orientation, ce qui permet d'orienter le cône de vision de la caméra[6]. Ainsi, l'utilisateur dispose d'un second point de vue – celui de la caméra virtuelle –, en plus de celui des yeux. Ce qu'observe ce deuxième « œil » est affiché dans un coin de la scène. On peut rapprocher ce système de la caméra endoscopique qui sert à explorer l'intérieur d'un corps humain, inaccessible directement par le chirurgien.

Pour manipuler directement le point de vue, il existe d'autres méthodes. Szalavari *et al.* [SG97] ont imaginé le système PIP pour Personal Interaction Pane qui se compose d'une tablette, d'un stylet et d'un casque de réalité augmentée.

[5] a. *camera in hand*
[6] a. *frustum*

Les objets de la scène sont superposés à la tablette et sont manipulés à l'aide du stylet. Soit la position de la caméra est celle du bout du stylet dans la scène et, dans ce cas, le nouveau point de vue est affiché sur la tablette. Soit la manipulation des propriétés du point de vue est réalisée par l'intermédiaire de widgets disposés sur la tablette. On trouve un système similaire dans [WDC99], où les auteurs utilisent un PalmPilot pour interagir sur l'environnement. En particulier, le point de vue est défini à l'aide du stylet du Palm, par l'intermédiaire de widgets. Par rapport au système PIP, cette méthode est utilisable aussi bien en réalité augmentée que virtuelle, le Palm prend moins de place que la tablette, et utilise un outil bien connu et maîtrisé.

Représentation miniature de l'environnement

Mackinlay *et al.* [MCR90] proposent de contrôler le point de vue de l'utilisateur à l'aide d'un périphérique tenu en main. Une fonction de mapping exponentielle est associée aux mouvements du périphérique afin d'amplifier les mouvements de la main de l'utilisateur. Pausch *et al.* [PB95] proposent d'utiliser une maquette virtuelle, représentation miniature du monde virtuel, que l'utilisateur tient dans sa main. C'est le monde en miniature[7] de Stoakley *et al.* [SCP95]. Le navigateur, déplace son propre avatar miniature dans le monde miniaturisé, et la mise à jour est automatiquement répercutée à l'échelle 1 : 1.

Elvins *et al.* [ENK97] proposent de fournir une représentation miniature[8] d'une partie du monde virtuel afin que l'utilisateur puisse facilement trouver son chemin. Leur travail repose principalement sur l'utilisation de repères visuels que l'utilisateur peut consulter *via* une maquette miniature d'une partie du monde. Par exemple, pour visiter une maison, au lieu d'avoir à parcourir toutes les pièces, il pourra utiliser un *worldlet* qui sera la représentation miniature d'une pièce particulière. Les *worldlets* peuvent être regroupés, comme dans un album de photographies, et on peut y accéder très facilement. Un *worldlet* ressemble à un Monde en miniature, mais au lieu de représenter toute la scène virtuelle, on n'en représente qu'une partie.

Techniques de mise à l'échelle du monde

Les objets virtuels sont souvent trop grands pour être manipulés directement, aussi certains auteurs ont proposé de les mettre à l'échelle de l'utilisateur, ce qui revient à modifier le point de vue. La mise à l'échelle peut donc être considérée comme un principe de navigation. Butterworth *et al.* [BAHO92] ont imaginé de réduire ou agrandir l'utilisateur dans la scène pour qu'il puisse manipuler un objet sans avoir à le modifier ou le déplacer. Robinett *et al.* [RH92] proposent une méthode similaire. Elle consiste à agrandir ou à réduire le monde relativement à l'utilisateur, comme dans Alice au Pays des Merveilles, jusqu'à ce que son bras se trouve au niveau de l'objet d'intérêt.

Slater *et al.* [SU94] présentent une alternative originale pour mettre l'environnement à l'échelle de l'utilisateur. Ce dernier appuie sur sa tête pour devenir plus petit, et pour grandir, il place sa main sous l'échine, puis fait un geste vers

[7] a. *World In Miniature*
[8] a. *worldlet*

le haut. Agrandir ou réduire directement le monde ou l'utilisateur est assez déstabilisant, car les points de repères qu'offre la scène ont tendance à disparaître, ou au contraire à prendre trop de place.

Mine *et al.* [MBJS97] imaginent une mise à l'échelle temporaire, dans le but de pouvoir manipuler localement des objets. L'utilisateur attrape le monde qui se rapproche de lui (figure 2.5). Il peut alors manipuler les objets d'intérêt localement à l'aide de ses mains. Lorsque la manipulation est terminée, le monde revient à sa taille initiale. Cette métaphore agit comme si l'utilisateur attrapait le monde pour le tirer à lui.

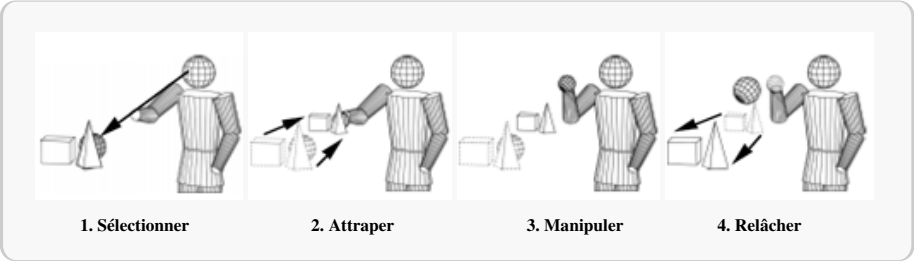


Fig. 2.5: Navigation par mise à l'échelle du monde.

Un bouton Pop Through a été utilisé par Zeleznik *et al.* [ZLJAFK02] pour se rapprocher d'un objet, d'une manière similaire à Mine *et al.* [MBJS97]. Le Pop Through est un bouton qui possède trois états : relâché, enfoncé, et à moitié enfoncé. Il est combiné avec un traqueur et placé sur une manchette que l'utilisateur enfile au bout de son index : c'est le Finger Sleeve. Ce périphérique est utilisé dans la métaphore du ZoomBack : en pointant un objet d'intérêt de son index, l'utilisateur appuie légèrement sur le Pop Through pour s'approcher de l'objet, voir s'il est intéressant ou non. Si oui, alors il appuie complètement et relâche, et il reste à présent à côté de l'objet. Si non, il relâche le bouton et reprend sa position initiale. Les auteurs proposent également une variante du ZoomBack, le LaserGrab, qui permet de se déplacer en orbite autour d'un objet.

Lentilles magiques

Les métaphores par lentilles magiques[9] sont des techniques de navigation qui permettent d'avoir un second point de vue sur une partie bien précise de la scène, en disposant en même temps d'une vue plus globale. Elles permettent donc de se focaliser sur une zone d'intérêt dans la scène virtuelle.

Stoev *et al.* [SSS02] présentent leur travail qui est inspiré par le Monde en miniature de [SCP95], et la Caméra dans la main de [WO90]. Ces métaphores posent un problème important : puisque l'utilisateur manipule son propre point de vue et que l'environnement virtuel est immédiatement mis à jour, il peut être désorienté. Avec les métaphores proposées par Stoev *et al.* [SSS02] l'utilisateur dispose d'un second point de vue qu'il peut valider après avoir vérifié

[9] a. *Through-The-Lens*

qu'il correspond à ce dont il a besoin. Pour cela, l'utilisateur tient dans sa main non dominante (gauche pour un droitier et droite pour un gaucher) une tablette semi-transparente. À sa surface est affiché le second point de vue miniature de la scène (figure 2.6). La main dominante manipule un stylet qui est suivi dans l'espace. Les auteurs proposent trois techniques différentes que nous allons détailler.

La première technique est appelée le Through-The-Lens Scene-In-Hand. Au début de la manipulation les deux points de vue sont identiques. L'utilisateur attrape simplement le second point de vue à l'aide de la pointe du stylet et le déplace selon ce qu'il souhaite regarder. Lorsqu'il estime être bien situé dans la scène, il relâche le second point de vue qui est affiché sur la tablette. Ce qui entraîne la modification du point de vue global de la scène. Il a donc manipulé la scène miniature en l'attrapant, et le point de vue global n'a été modifié qu'une fois la manipulation validée.

Le Through-The-Lens World-In-Miniature offre une vue miniature de la scène virtuelle, à la manière du Monde en miniature. Mais la différence est que le l'affichage du monde miniature est fait de telle manière que l'utilisateur a l'impression de regarder à travers un judas : la scène semble être affichée sous la tablette qu'il tient dans sa main non dominante. Il a l'impression d'observer le monde en miniature à l'intérieur d'une fenêtre. Pour modifier le point de vue, il lui suffit d'encadrer une zone du second point de vue, directement sur la tablette. Cette zone sera mise à l'échelle pour couvrir tout le champ de vision du second point de vue ; il s'agit donc d'un zoom. Le point de vue global est alors changé pour correspondre à celui affiché sur la tablette.

Enfin, la dernière méthode présentée par les auteurs s'intitule le Through-The-Lens Eyeball-In-Hand. Cette fois-ci, le bout du stylet agit comme une caméra que l'utilisateur balade dans la scène. La vue affichée sur la tablette change en conséquence et lorsque le bon point de vue est trouvé, l'utilisateur valide son choix, ce qui entraîne une mise à jour automatique du point de vue global.

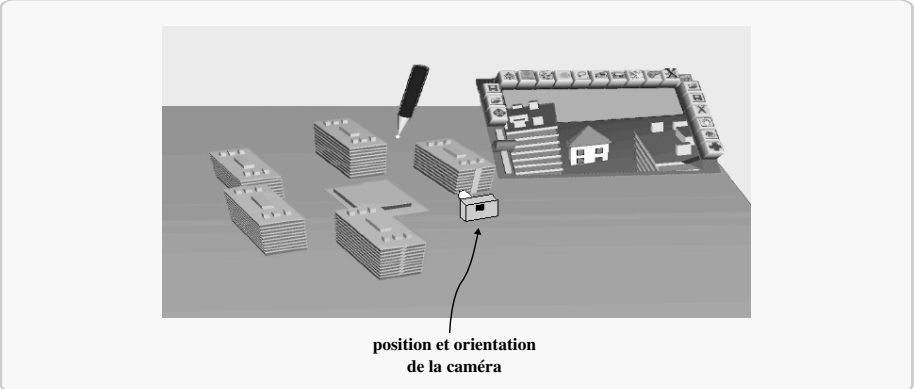


Fig. 2.6: Métaphore *through-the-lens* - l'utilisateur manipule le point de vue à l'aide d'un stylo, lequel est affiché sur la tablette. Il a l'impression de voir à travers une fenêtre le nouveau point de vue.

2.3.3 Navigation par indication de la direction

Définition Les techniques d'indication de la direction consistent à orienter le déplacement en indiquant un but à atteindre.

Un être humain se déplace à l'aide de repères qui lui permettent de s'orienter dans l'espace. Il est donc possible de se mouvoir en indiquant une direction à suivre. Parmi les techniques d'interaction reposant sur cette méthode, il y a celles qui exploitent le regard et celles qui utilisent le pointage de l'index ou la direction du torse, ou de manière plus générale l'orientation du corps.

Direction du regard

Mine *et al.* [Min95] introduisent une technique de déplacement relativement simple où la direction est donnée par le regard : l'utilisateur se déplace dans la direction $\vec{d}$ vers laquelle il regarde. Plus précisément, un traqueur suit les mouvements de sa tête afin de déterminer le sens du déplacement. L'utilisateur peut ensuite se déplacer le long de ce vecteur, dès lors qu'il est normalisé. Pour démarrer et stopper le déplacement, il suffit d'appuyer sur un bouton, et un joystick permet de faire varier la vitesse v. À chaque instant la nouvelle position p de l'utilisateur se calcule selon l'équation 2.1 :

$$p_{courante} = p + v\frac{\vec{d}}{||\vec{d}||} \tag{2.1}$$

Cette métaphore d'interaction s'avère en pratique assez facile à comprendre et à contrôler. Par contre, son principal inconvénient réside dans le fait qu'il est impossible de se déplacer dans une direction et regarder ailleurs en même temps. Le déplacement peut en outre se révéler assez désagréable puisque la tête n'est jamais parfaitement droite. En effet, les yeux se réorientent en permanence pour pallier aux mouvements parasites du reste de la tête. Enfin, l'amplitude des mouvements du cou est limitée ; par exemple, il sera délicat pour l'utilisateur de regarder derrière soi sans bouger son corps, ou lever la tête pour se déplacer verticalement.

Direction pointée du doigt

Nous venons de voir que les déplacements dirigés par le regard ne permettent pas d'utiliser la vue pour faire autre chose. Bowman *et al.* [BKH97] ont montré que la navigation par pointage est plus efficace. La direction du regard et la direction du déplacement peuvent donc être différentes. Le plupart du temps, un traqueur est placé sur la main de l'utilisateur, et lorsque celui-ci pointe son doigt en direction d'un endroit (vecteur de direction $\vec{d}$), il peut ensuite se déplacer le long de ce vecteur dès lors qu'il est normalisé. Un concept analogue existe dans les jeux vidéo, où le joueur avance à l'aide du clavier tandis qu'il oriente son regard dans n'importe quelle direction à l'aide de la souris. Pour débuter ou arrêter le voyage, il est possible de recourir à un bouton, à un geste de la main, ou encore aux Pinch Gloves de Bowman *et al.* [BKH97, BWHA98].

Mine *et al.* [MBJS97] ont proposé une extension intéressante à cette technique. En effet, il est possible d'utiliser simultanément les deux mains pour déterminer à la fois un vecteur de déplacement et une vitesse associée à cette direction. Pour cela, il suffit de connaître la position des deux mains dans l'espace. Pour calculer la vitesse, on calcule la distance entre les deux mains : plus l'écartement est faible et plus le déplacement sera lent. En pratique, cette technique est plus compliquée à utiliser que la précédente. En effet, il y a plusieurs aspects qui rentrent en ligne de compte. Bien que plus flexible, elle s'avère également plus complexe à utiliser. Par exemple, lorsque l'utilisateur souhaite ralentir sa progression dans une scène virtuelle, il rapproche ses deux mains l'une vers l'autre, ce qui diminue sa vitesse, mais qui peut également modifier légèrement l'orientation du vecteur. En outre, les deux mains sont utilisées en même temps, ce qui entraîne une charge cognitive supplémentaire, par rapport à l'utilisation d'une seule main.

Direction donnée par le torse

La direction du regard ne permet pas de regarder ailleurs lors du déplacement, et le pointage avec le doigt et les mains ne permet pas de faire une autre tâche manuelle en même temps que l'utilisateur voyage. À l'inverse, lorsque la direction est celle du torse[10], comme le proposent Bowman *et al.* [BKH98], il est à la fois possible de se déplacer en regardant ailleurs et en réalisant une tâche manuelle quelconque. Comme on a une tendance naturelle à tourner le torse dans la bonne direction quand on marche vers un certain endroit, le vecteur direction est déterminé à l'aide d'un traqueur, porté la plupart du temps sur la poitrine. Il faut cependant veiller à ne pas le positionner trop haut sur le torse. En effet, les mouvements de la tête (par exemple lorsque l'utilisateur souhaite regarder ailleurs pendant qu'il se déplace) peuvent engendrer de légères rotations du haut du torse non voulues.

Cette technique présente quelques inconvénients majeurs. Il est par exemple assez délicat de tourner le torse vers le ciel ou le sol pour monter ou descendre ; les déplacements dans le sens du torse sont donc limités au plan horizontal.

Contrôle virtuel du mouvement

Pour indiquer la direction, Wells *et al.* [WPA96] ont imaginé un tapis circulaire d'environ un mètre de diamètre dont la surface est parsemée de capteurs : le Virtual Motion Controller (figure 2.7) du Hit Lab de l'Université de Washington. Dans un premier temps, il détecte des variations de poids sur sa surface, puis il détermine la direction souhaitée par l'utilisateur. Ce tapis a une forme circulaire et la distance par rapport au centre donne la vitesse courante de déplacement. Pour avancer, l'utilisateur ne marche pas ; il se penche simplement en avant dans la direction qu'il souhaite prendre.

Ce tapis pose un certain nombre de problèmes. Premièrement, il est délicat de rester au centre sans bouger. Imaginons, par exemple, que l'on soit légèrement déséquilibré en arrière, on commencera à reculer sans forcément le vouloir. En-

[10] a. *torso-directed*

Fig. 2.7: Tapis à capteurs de poids *Virtual Motion Controller*, Hit Lab Washington.

suite, il est difficile de monter ou descendre à l'aide de ce tapis. Les mouvements se situent avant tout dans le plan horizontal.

Techniques de vol

Pour certains auteurs, la marche est inadaptée à l'exploration d'un environnement virtuel. Ils préconisent de parcourir la scène en volant. Cependant, Usoh *et al.* [UAW+99] ont réalisé une étude comparative entre le vol, la marche sur place et la marche réelle. Ils observent que la sensation de présence est plus forte pour la marche sur place que pour le vol, et plus encore pour la marche réelle.

Butterworth *et al.* [BAHO92] préconisent la solution du vol dans son modeleur 3DM. La direction est donnée en orientant un périphérique à six degrés de liberté dans l'espace. Les auteurs proposent également d'attacher un curseur virtuel (lié aux mouvements du périphérique) au monde synthétique ; l'utilisateur tire alors l'environnement à lui, en bougeant le curseur, jusqu'à obtenir le point de vue qui lui convient.

Pour Robinett *et al.* [RH92], le vol se définit comme une opération de translation dans la direction pointée par le périphérique tenu en main. Ils précisent que pour que l'utilisateur ait la sensation de bouger dans le monde virtuel, il est plus simple de modifier le point de vue de l'utilisateur que de modifier tout l'environnement. La direction peut également être déterminée par le regard [SU94, SUS95, UAW+99].

L'utilisation des mains dans les métaphores de vol est courante. Mine *et al.* [MBJS97] imaginent une métaphore basée sur l'utilisation conjointe des deux mains (interaction bi-manuelle), qui forment un vecteur. La direction du vecteur donne la direction du vol, et sa longueur fournit la vitesse.

2.3.4 Navigation par accroche

Définition Lorsque l'utilisateur attrape un objet fixé au monde virtuel, ou la scène elle-même (par exemple en fermant le poing), il lui est possible de se déplacer en éloignant ou en rapprochant ce point d'accroche de son corps.

La manipulation par accroche est un type d'interaction qui repose sur l'utilisation d'une ou deux mains pour modifier le point de vue en rapprochant ou en éloignant tout ou partie de la scène de soi. Il est possible d'attraper le monde virtuel, ou un objet seulement de cet environnement. Lorsque le déplacement est conséquent, il devient nécessaire de recommencer plusieurs fois l'opération d'accroche.

Attraper le monde

Ware *et al.* [WO90] ont imaginé que l'on pouvait attraper l'environnement virtuel qui nous entoure pour modifier notre point de vue. De fait, le monde se comporte alors comme un seul et unique objet que l'on approche ou que l'on éloigne de ses yeux, pour en observer un coin particulier. Il suffit de fermer le poing pour que le monde se retrouve attaché à la main. À l'inverse lorsque la paume est ouverte, le monde est alors détaché de la main et reste figé à l'endroit même où l'utilisateur l'a relâché.

Cette technique est assez simple à développer. Cependant, pour éviter de désorienter l'utilisateur, il ne faut pas prendre en compte les rotations de sa main. En outre, si l'endroit visé est assez éloigné dans la scène, il faudra faire un certain nombre de mouvements pour attraper, tirer, puis relâcher le monde, pour arriver à destination. Pour atténuer cet inconvénient, Mapes *et al.* [MM95] proposent d'améliorer l'interaction en utilisant conjointement les deux mains, plutôt qu'une seule. Le mouvement pour amener le monde à soi, ou au contraire l'éloigner, s'apparente à celui que l'on fait lorsque l'on tire sur une corde. Les auteurs proposent également d'autoriser les rotations du monde : une main devient le pivot et l'autre détermine la rotation à effectuer. Cette technique monopolise les deux mains en même temps. Il est donc impossible de réaliser une autre manipulation pendant le déplacement. Par contre, il est tout à fait envisageable de bouger verticalement, et non pas uniquement dans le plan horizontal.

Attraper un objet fixe

En lieu et place d'attraper le monde, il est également possible d'agripper un objet. D'ordinaire c'est lui qui bouge lorsqu'on le tire vers nous, mais si l'objet est fixé dans l'univers virtuel, alors on pourra se rapprocher de lui. Plusieurs techniques ont été imaginées pour cela. La plus simple consiste simplement à utiliser la main réelle de l'utilisateur, ce qui marche bien dans un environnement de réalité augmentée. Par contre, dans des dispositifs de réalité virtuelle, des phénomènes d'occlusion peuvent apparaître. On peut alors utiliser un avatar virtuel de la main réelle qui se portera à hauteur de l'objet considéré. Dans la réalité, ce type de déplacement peut s'apparenter à un pompier qui s'agrippe à une corde pour se hisser. Dans cet exemple, nous voyons que l'objet corde est fixe et c'est l'utilisateur qui bouge.

Pour simplifier la sélection de l'objet, Pierce *et al.* [PFC+97] ont imaginé la technique de sélection dans le plan image[11]. L'idée est de projeter le monde $3D$ dans un plan $2D$, puis de sélectionner l'objet à attraper en pointant par exemple le doigt dessus. Le système considère alors le vecteur passant par les yeux et le doigt pour déterminer les objets qui sont en intersection. Lors d'une utilisation classique des techniques de plan-image (Head Crusher, Sticky Finger et Lifting Palm), c'est l'objet qui se rapproche de l'utilisateur, alors qu'en mode navigation, les objets restent fixes, et c'est le point de vue qui se déplace vers eux. Le système basé sur les poupées vaudou de Pierce *et al.* [PSP99] fonctionne sur le même principe de sélection des objets par plan image, mais cette fois sur des miniatures des objets auxquels l'utilisateur souhaite s'accrocher. Nous détaillerons plus loin ces techniques d'interaction.

2.3.5 Navigation planifiée

Définition La navigation planifiée est une autre façon de se déplacer dans une scène virtuelle, le long d'un chemin prédéterminé, en indiquant la route à suivre. À partir de là, le point de vue de l'utilisateur est modifié de manière à suivre la route, sans que celui-ci ait à intervenir.

Ce type de navigation peut s'apparenter au déplacement d'une caméra lors du tournage d'un film[12]. En effet, beaucoup de scènes sont prévues à l'avance, avec des vues sous des angles particuliers. Les techniques de navigation planifiées offrent comme avantage de ne pas se soucier du déplacement, contrairement aux techniques dirigées par exemple. Elles laissent moins de degrés de liberté, mais la charge cognitive qui y est associée, se trouve également réduite.

Chemin dessiné

Pour planifier un chemin il est possible de le dessiner préalablement. Igarashi *et al.* [IKMT98] proposent une technique d'interaction, où l'utilisateur dessine la route souhaitée à l'aide d'un pointeur comme par exemple un stylo. Le système calcule automatiquement les mouvements de l'avatar le long du chemin, en projetant le tracé dessiné à l'écran, sur la surface $3D$ dans le monde virtuel. La direction de l'avatar est fixée à la tangente de la courbe projetée.

Cette projection est réalisée de manière assez intelligente ; si l'utilisateur dessine un chemin qui passe par un tunnel dont il ne peut voir l'intérieur, le système détermine la bonne projection. À l'inverse, lorsque le chemin doit passer au dessus d'un ravin, l'algorithme adapte automatiquement le chemin $3D$ pour éviter de tomber au fond du gouffre. Des mécanismes d'évitement d'obstacles, ou au contraire d'ouvertures dans des obstacles (créer un tunnel temporaire dans une montagne pour la traverser) ont également été testés avec succès.

Une extension de ce système peut être réalisée à l'aide de cartes. En effet, lorsque l'environnement virtuel est grand, l'utilisateur ne peut pas forcément atteindre, de façon précise, tous les endroits. Par contre, il lui est tout à fait possible de tracer un chemin sur une carte $2D$ ou $3D$. Le système transforme

[11] a. *Image-Plane*
[12] a. *travelling*

les coordonnées de la courbe sur la carte, en coordonnées dans le monde virtuel.

Galyean *et al.* [Gal95] proposent de contraindre les mouvements de l'utilisateur en lui laissant une certaine marge de manoeuvre, c'est-à-dire en combinant navigation guidée et navigation libre. L'utilisateur se trouve dans un véhicule qui suit un chemin (par exemple un bateau sur une rivière). Il peut s'écarter du bateau lorsqu'il souhaite voir de plus près les animaux sur les berges, mais il reste attaché à l'embarcation par un ressort qui le tire en arrière. En outre, le ressort est d'autant plus raide que le courant est fort.

Points de passage

Lorsque le chemin à parcourir ne s'intéresse qu'à certains endroits particuliers, il est possible de n'indiquer que certains points de passage. Le système détermine ensuite un chemin continu passant par ces points, sans que l'être humain ait à intervenir. Bowman *et al.* [BDHN99] utilise ces concepts pour tester la capacité des utilisateurs à s'orienter mentalement dans un espace confiné (corridor). Les auteurs ont montré que lorsque les utilisateurs peuvent voir le chemin qui passe par les points de passage (directement dans l'environnement $3D$ ou bien sur une carte), ils ont une meilleure compréhension mentale du monde virtuel. Précisons que dans cette implantation, la carte était en $3D$, puisque le corridor à explorer pouvait tourner à droite ou à gauche et monter ou descendre.

Cartes et cibles

La navigation orientée peut être réalisée à l'aide de cartes. Leur utilisation en environnement virtuel favorise la connaissance spatiale, en favorisant la construction de la carte cognitive d'un utilisateur. Pour Edwards *et al.* [EH97], l'utilisation des cartes en environnement virtuel maintient un modèle cognitif plus cohérent de l'environnement global lors de la navigation. En effet, une carte permet de mieux appréhender la topologie d'un environnement grâce à une représentation exocentrique de l'observateur par rapport à son environnement.

L'affichage d'une carte dépend à la fois de la position de l'utilisateur sur la carte – est-il centré en permanence sur la carte ou non, et de l'orientation de cette dernière par rapport au Nord géographique. On trouve deux types d'orientation : les cartes *nord-dépendantes* et les cartes *vue-dépendantes*. Dans le premier cas, c'est la connaissance globale de l'environnement qui est avantagée. Cependant, l'utilisateur doit être capable d'effectuer une rotation mentale pour associer mentalement les éléments représentés sur les cartes avec ceux de l'environnement. Dans le second cas, les cartes dépendantes du point de vue favorisent au contraire l'association directe entre les éléments de l'environnement et ceux de la carte, puisque l'orientation de la carte correspond en permanence à la direction du regard de l'utilisateur dans la scène.

Selon la manière dont l'utilisateur se repère dans son environnement, la construction de la carte cognitive sera différente. Certaines personnes utilisent essentiellement une stratégie égocentrique – centrée sur eux-mêmes, alors que d'autres préfèrent une stratégie exocentrique – centrée sur les objets qui les entourent. Dans ce cas, une carte dépendante du point de vue est favorable.

Bowman *et al.* [BDHN99] essayent de mesurer la capacité des utilisateurs à s'orienter après avoir traversé un corridor $3D$. À la fin du parcours, ils doivent indiquer dans quelle direction se trouvent certains des objets qu'ils ont rencontrés pendant le trajet. Les auteurs constatent que les utilisateurs emploient une ou plusieurs des six stratégies suivantes :

- s'arrêter et regarder ;
- pointage proprioceptif : l'utilisateur pointe dans la direction des objets qu'il a déjà regardée ;
- retour en arrière : l'utilisateur se retourne et recule lorsqu'il arrive à la fin du corridor ;
- refaire le chemin : l'utilisateur revient sur le chemin avant de continuer à avancer, pour le mémoriser ;
- bouger à travers les murs ;
- voir le corridor du dessus.

Ils constatent que l'utilisation d'une carte des corridors $3D$ fait baisser les capacités d'orientation. Ils émettent alors l'hypothèse que cette baisse provient d'une charge cognitive supplémentaire.

Darken *et al.* [DC99] ont également travaillé sur les cartes. Ils pensent que l'utilisation de cartes virtuelles peut avoir un impact positif sur les performances lors du processus de navigation. Les plus performantes sont celles qui s'orientent automatiquement en fonction de la direction que prend l'utilisateur. Une carte virtuelle apporte des avantages par rapport à une carte réelle, puisqu'elle peut être interactive : elle peut par exemple indiquer la position courante de l'utilisateur et afficher un certain nombre de données dynamiques (météorologie, danger, ...). Bien qu'il soit possible d'afficher des informations supplémentaires, il est également possible d'en enlever pour rendre la lecture de la carte plus simple et diminuer la charge cognitive nécessaire. L'expérience joue un rôle indéniable, puisque ceux qui savent bien se servir de leur carte s'orientent plus facilement et plus rapidement que les autres. Les auteurs ont déterminé trois principes d'utilisation d'une carte virtuelle :

- pour les tâches ciblées, une carte auto orientée est préférable ;
- pour des recherches naïves (découverte d'un lieu par exemple), une simple carte orientée vers le Nord est souhaitable ;
- les utilisateurs qui ont de bonnes capacités spatiales seront capables de se servir de n'importe quel type de carte.

Les méthodes pour afficher et manipuler les cartes sont nombreuses. Parmi les techniques les plus utilisées, on trouve notamment les cartes qui se manipulent par l'intermédiaire de widgets, ou encore l'utilisation de PDA (PalmPilot, ...) et d'un stylo.

2.4 La sélection

L'interaction avec des objets virtuels – comme par exemple déplacer un objet – nécessite de les avoir préalablement sélectionnés. L'utilisateur doit être en mesure d'indiquer sur quelle(s) cible(s) la manipulation aura lieu. Cette tâche constitue ainsi la base de tout système interactif, et se décompose en deux parties caractéristiques :

- la désignation : un mécanisme dédié à l'identification du ou des objets à sélectionner ;
- la validation : un signal ou une commande pour confirmer le choix courant de la sélection, c'est-à-dire le ou les objets préalablement désignés.

Dans nos environnements de bureau, nous avons pris l'habitude de réaliser successivement ces deux sous-tâches. Nous survolons par exemple une icône qui change alors de couleur pour indiquer qu'elle est potentiellement sélectionnable, puis nous cliquons dessus pour lancer une commande particulière ou ouvrir un fichier. Désignation et validation sont ici explicites. À l'inverse, il peut arriver que ces deux sous-tâches ne forment qu'une seule entité, où la validation est totalement implicite. En reprenant notre exemple précédent, l'utilisateur passe simplement son pointeur sur l'icône sans avoir besoin d'appuyer sur un quelconque bouton.

La sélection rassemble de nombreuses techniques de désignation / validation, selon que les objets de la scène virtuelle sont proches ou distants, petits ou grands, qu'il y en ait un ou plusieurs à manipuler, etc. Nous allons détailler dans cette section les techniques de sélection locale et distante, les métaphores de sélection dirigée, l'emploi de commandes vocales et enfin la sélection dans une liste de choix.

2.4.1 Sélection locale

Définition Une sélection est dite locale lorsque l'objet que l'utilisateur désire sélectionner se trouve à portée directe de sa main, sans aucun déplacement dans la scène virtuelle.

De façon similaire à ce que nous faisons dans la réalité, il apparaît naturel et simple d'attraper un objet pour le manipuler. La réalité virtuelle permet bien sûr le contact direct entre la main réelle de l'utilisateur et un objet virtuel, mais elle offre également des solutions impossibles dans notre quotidien. Par exemple, il est possible de cloner les objets et de travailler sur des miniatures de tout ou partie de la scène virtuelle.

Sélection par préhension

Lorsque l'utilisateur emploie sa propre main pour sélectionner et manipuler les objets de la scène virtuelle de Sturman *et al.* [SZP89], on parle de contact par préhension. Une partie de la main doit entrer en contact avec l'objet à sélectionner, pour préalablement le désigner. Ensuite, plusieurs mécanismes peuvent intervenir pour valider la sélection : presser un bouton, laisser la main un certain temps en contact avec l'objet, etc. La validation peut également être implicite

dès que la main entre en contact avec l'objet.

Le principal avantage est que cette technique est intuitive et parfaitement naturelle. Par contre, la distance de manipulation est limitée, les objets de grande taille sont difficiles à manipuler, et l'occlusion de la scène par la main pose problème.

Comme pour la plupart des autres métaphores que nous allons voir plus loin, le retour visuel de la sélection par préhension aide l'utilisateur à connaître quel est l'état de l'objet qu'il souhaite sélectionner. L'objet peut être non sélectionné (état normal), désigné (sélection non encore validée) ou sélectionné (sélection validée). Par exemple, un changement de la couleur de l'objet permet d'indiquer qu'il est désigné ou sélectionné.

Sélection dans un monde miniature

Les environnements de réalité virtuelle sont faits pour apporter des nouveautés par rapport au monde réel. En particulier, la réalité virtuelle peut apporter à l'utilisateur plus qu'un unique point de vue du monde virtuel à l'échelle 1 : 1. À cette fin, Stoakley *et al.* [SCP95] ont proposé de superposer un second monde virtuel, qui est la réplique en miniature [13] de celui à l'échelle 1 : 1 (figure 2.8). Dans l'implantation des auteurs, une modification sur un des deux mondes entraîne des modifications identiques et instantanées à l'autre échelle.

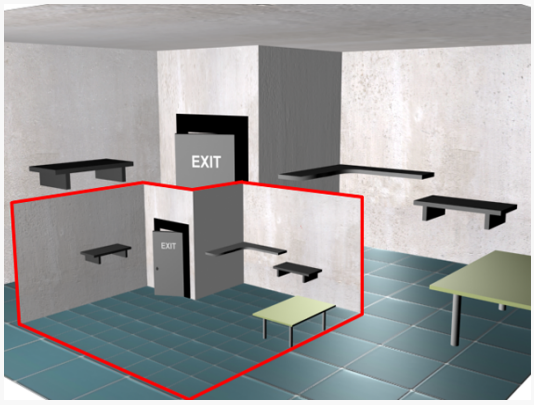


Fig. 2.8: Vue du monde miniature superposé au monde virtuel 1 : 1.

La main non dominante (gauche pour un droitier et droite pour un gaucher) tient une planchette munie de capteurs de position et d'orientation afin de suivre les déplacements de la maquette dans l'espace. Le monde virtuel en miniature (une partie du monde virtuel global) est représenté comme une maquette tridimensionnelle sur la tablette. La main dominante (droite pour un droitier et gauche pour un gaucher) tient une balle de tennis munie de deux boutons et d'un capteur de position. Le premier bouton est utilisé pour la sélection des

[13] a. *world in miniature* ou *WIM*

objets, et le second pour effectuer une action spécifique dans l'application.

La métaphore du monde miniature est assez complexe. Elle est utilisable aussi bien pour la sélection, pour la manipulation, que pour la navigation. Elle est très utile pour déplacer des objets volumineux, ou hors de portée directe sans déplacement de l'utilisateur dans la scène.

Dans un environnement de type plan de travail virtuel – par exemple le *Workbench*, le bras de l'utilisateur peut venir cacher tout ou partie de la scène miniature (phénomène d'occlusion). Il s'avère alors important de bien choisir le meilleur emplacement du monde miniature dans la scène globale, afin que le bras de l'utilisateur (ou une autre partie de son corps) ne gène pas son utilisation.

Sélection par poupées vaudou

La technique du monde miniature fournit à l'utilisateur une réplique miniaturisée du monde virtuel global, qu'il peut manipuler pour affecter le monde de taille réelle. Cependant, miniaturiser le monde entier pour qu'il tienne dans une main pose des problèmes, comme la difficulté de sélectionner les petits objets et de les déplacer précisément dans l'espace.

Pierce *et al.* [PSP99] ont proposé la technique dite des poupées vaudou[14] pour pallier ces limitations. Il s'agit d'une technique d'interaction à deux mains, avec laquelle l'utilisateur crée ses propres parties miniatures de l'environnement, qui sont appelées des poupées. La poupée est tenue dans la main non dominante de l'utilisateur, alors que la main dominante permet de faire une action spécifique sur l'objet miniaturisé.

La sélection des objets à miniaturiser se réalise à l'aide de plusieurs techniques : affichage d'un curseur (figure 2.9), utilisation d'un cadre de sélection, ou encore un cercle englobant. Elle permet de travailler à des échelles multiples sans se soucier du redimensionnement proprement dit. En effet, quel que soit l'objet, ou le groupe d'objets sélectionné, la miniature correspondante est toujours affichée à une taille constante. Elle autorise à la fois les objets visibles et cachés à être manipulés. En effet, il est possible d'orienter la miniature dans l'espace, de manière à ce que les objets cachés deviennent visibles.

La sélection par poupées vaudou s'avère être une bonne variante au monde miniature. Ainsi, au lieu de travailler par rapport au référentiel du monde, l'utilisateur a tout loisir de changer le cadre de référence pour celui qui lui convient.

2.4.2 Sélection distante

Définition Une sélection distante correspond à la situation où l'utilisateur se trouve hors de portée physique de l'objet qu'il désire sélectionner.

Les objets virtuels ne peuvent pas toujours être attrapés directement, car ils se situent trop loin de l'utilisateur. Il est bien entendu possible de se déplacer, puis de sélectionner l'objet pour travailler localement dessus. Mais dans ce

[14] a. *voodoo dolls*

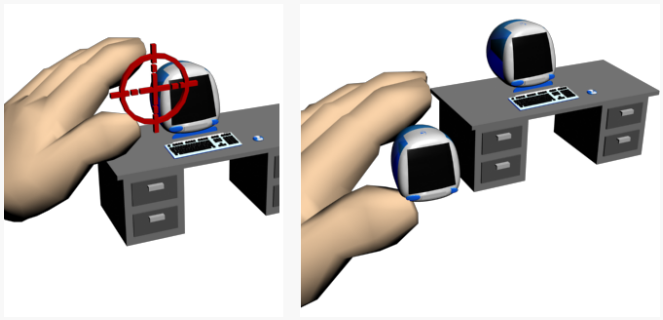


Fig. 2.9: Sélection par poupée vaudou sous forme de curseur.

cas, il faudra éventuellement retourner à sa position d'origine ; de plus, en s'approchant d'un objet, il arrive que l'on perde une partie de la vue sur la scène virtuelle.

Certains auteurs proposent d'utiliser un intermédiaire visuel pour attraper le ou les objets d'intérêt. Ce peut être un simple curseur $3D$ ou encore un rayon virtuel dont la forme peut varier. Nous allons voir qu'il est également possible d'allonger une représentation virtuelle du bras de l'utilisateur, c'est-à-dire un avatar, afin que celui-ci vienne se porter à hauteur de l'objet.

Sélection par curseur $3D$

L'utilisateur contrôle un curseur $3D$ par un moyen quelconque (par exemple avec un joystick). La sélection devient effective lorsque le curseur entre en contact avec un objet. Si le curseur est attaché virtuellement à la main de l'utilisateur, cette métaphore est alors très proche de la sélection par contact. La figure 2.10 de gauche montre un utilisateur qui manipule un joystick pour déplacer le curseur $3D$ dans l'espace. Dès qu'il se trouve en contact avec le curseur $3D$ (figure 2.10 de droite), l'objet est entouré d'une sphère filaire englobante pour indiquer qu'il est à présent sélectionné.

Fig. 2.10: L'utilisateur dirige le curseur $3D$ vers l'objet (à gauche) qui entre en contact avec l'objet (à droite).

Sélection par lancer de rayon

La technique dite du lancer de rayon[15], également nommée rayon virtuel, a été introduite par Bolt [Bol80], puis a ensuite été reprise et améliorée par de nombreux auteurs comme [MCR90, Min95, ZBM94]. Dans son implantation, Mine [Min95, Min96] utilise une métaphore dérivée de la technique direction pointée du doigt. Un rayon lumineux virtuel part de la main de l'utilisateur, et les objets sont sélectionnés lorsque le rayon laser entre en contact avec eux. Le ou les objets sélectionnés se retrouvent alors attachés au bout du rayon pour la manipulation (figure 2.11).

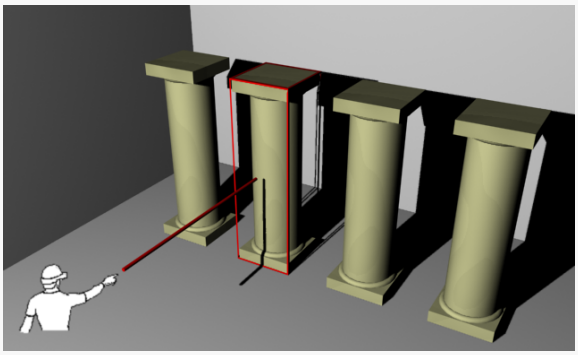


Fig. 2.11: Technique du lancer de rayon.

Zhai *et al.* [ZBM94] ont étendu le lancer de rayon simple à un rayon au bout duquel est attachée une croix tridimensionnelle semi transparente – le Silk Cursor. Cette métaphore est basée sur la loi de Fitts, qui dit que le temps de sélection diminue lorsque la surface à sélectionner augmente en $2D$. Les auteurs ont appliqué cette loi à un environnement $3D$ et un volume de sélection : plus le volume de sélection est grand, plus le temps de sélection diminue. Alors que la transparence permet de ne pas avoir d'occlusion, l'information volumique sert notamment à mieux situer le rayon dans l'espace. L'information de profondeur est importante, mais se révèle, dans la pratique, nettement moins performante sans ombrage. En effet, Zhai *et al.* ont remarqué qu'il est plus délicat d'apprécier la distance entre les objets sans cette dernière information. Les auteurs conseillent donc d'ajouter des ombres aux objets de la scène, ainsi qu'au Silk Cursor. Nous pouvons ajouter que pour presque toutes les autres métaphores d'interaction, l'information de l'ombre permet de mieux apprécier les distances et la profondeur dans la scène virtuelle. De Amicis *et al.* [DAFS01] utilisent la même métaphore que le Silk Cursor, mais la croix est remplacée par un volume sphérique, qui est attaché au bout d'un stylo physique que l'utilisateur tient en main.

Les métaphores du rayon laser et du Silk Cursor souffrent de quelques limitations importantes. Il est impossible de sélectionner un objet caché derrière un autre. Le centre de la manipulation (par exemple pour une rotation) est

[15] a. *ray-casting*

toujours la main réelle de l'utilisateur, ce qui est particulièrement limitant pour tourner un objet situé assez loin dans la scène. En outre, rapprocher ou éloigner un objet s'avère contraignant : l'utilisateur doit reposer et reprendre l'objet régulièrement avec le rayon laser.

Olwal *et al.* [OF03] ont imaginé le pointeur flexible (figure 2.12) qui est une extension du lancer de rayon classique, car il permet de pointer plus facilement des objets cachés en tout ou partie par d'autres objets. En effet, le rayon peut se tordre dans l'espace. La courbure et la longueur du rayon sont contrôlées à l'aide des deux mains, et la courbe ainsi créée est une courbe de Bézier avec 3 points de contrôle.

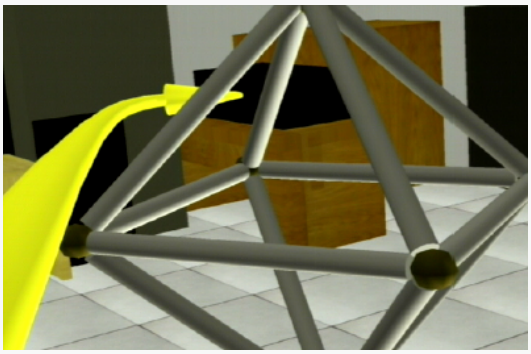


Fig. 2.12: Le pointeur flexible : sélection à l'aide d'un rayon laser flexible.

Sélection par projecteur

La sélection par projecteur[16] est une technique similaire au lancer de rayon classique (cf. paragraphe précédent), imaginée par Liang *et al.* [LG93], avec une petite modification : le rayon est conique (figure 2.13). Ce qui a pour avantage de pouvoir sélectionner les objets très petits et lointains. Les objets paraissent de plus en plus petits à mesure que la distance à l'utilisateur augmente. La solution retenue consiste à rendre le rayon de plus en plus large.

Le rayon conique peut poser des problèmes dans le cas où la scène contient plusieurs objets de petite taille, et qui sont assez éloignés de l'utilisateur. Si l'utilisateur désire sélectionner un objet parmi un groupe d'objets, il risque d'en sélectionner plusieurs en même temps.

Sélection par ouverture

Forsberg *et al.* [FHZ96] ont présenté une technique qui permet d'atténuer le défaut de sélection des petits objets avec la métaphore de sélection par projecteur (cf. paragraphe précédent). Les auteurs nomment leur technique la sélection par ouverture[17], puisque c'est l'utilisateur qui fait varier l'ouverture du cône se-

[16] a. *spotlight*
[17] a. *aperture based selection*

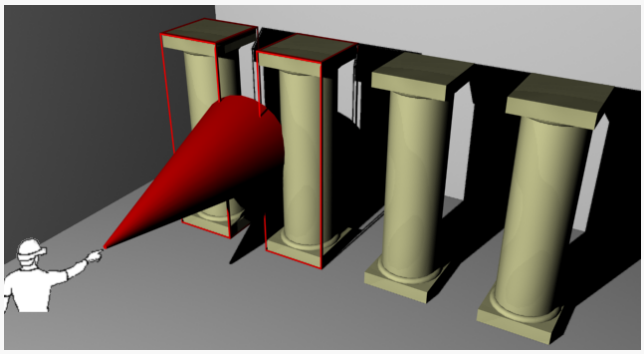


Fig. 2.13: Sélection par projecteur.

lon ses besoins. Elle utilise le principe de la mire, c'est-à-dire que le rayon part de l'œil dominant de l'utilisateur et passe par l'index (figure 2.14). Il y a ainsi deux manières de bouger le rayon conique : soit bouger la main, soit bouger la tête.

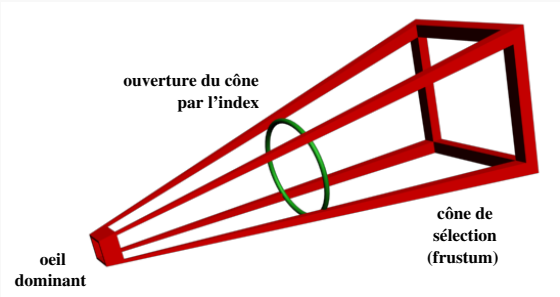


Fig. 2.14: Sélection par rayon à ouverture variable.

Chacun d'entre nous possède un œil dominant : cet œil est privilégié par le cerveau pour le traitement de l'information. Pour le trouver, il suffit de tendre le bras, puis de fixer du regard un doigt en fermant alternativement les deux yeux. Le doigt bougera davantage dans l'espace avec l'un des deux yeux fermés ; cet œil est l'œil dominant.

Par rapport à la sélection par projecteur, la sélection par ouverture est moins sensible au bruit induit par les tremblements de la main. Elle est cependant plus lourde à mettre en œuvre : d'une part, il est nécessaire de déterminer quel est l'œil dominant, et d'autre part, il faut disposer de deux capteurs de position et d'orientation : un pour suivre la tête et un pour suivre la main.

Sélection par ouverture avec orientation

Forsberg [FHZ96] ont présenté une variante à la sélection par ouverture (cf. paragraphe précédent). Dans le monde réel, avant de manipuler un objet, nous

orientons nos mains pour qu'elles correspondent à l'orientation de cet objet. Par exemple, nous orientons nos mains pour attraper une tasse de thé ou un livre.

On peut utiliser de façon similaire l'information sur l'orientation tridimensionnelle fournie par le capteur de la main pour sélectionner certains objets. Avec cette méthode, si plusieurs objets tombent dans le cône de sélection orienté, seul celui dont l'orientation est la plus proche de celle du capteur est choisi. La figure 2.15 montre A) l'orientation qui sera employée pour sélectionner le tore et B) l'orientation qui sera utilisée pour sélectionner le cylindre. La figure 2.16 montre le volume de sélection défini à l'aide d'un cercle qui est orienté dans l'espace par l'utilisateur.

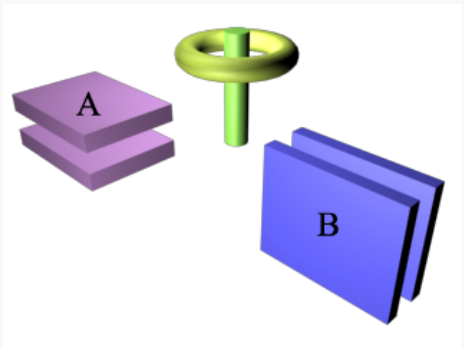


Fig. 2.15: Principe de correspondance orientation cône orientation objet.

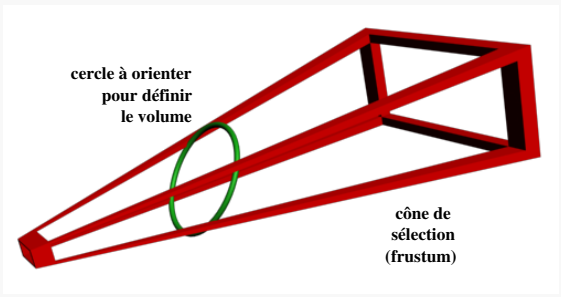


Fig. 2.16: Sélection par ouverture avec orientation.

Sélection par extension de la main

Poupyrev *et al.* [PBWI96] ont imaginé une technique utilisant une main virtuelle dans l'espace qu'ils nomment le *GoGo*[18]. Le constat initial était qu'une main virtuelle classique, qui imite parfaitement les mouvements de la main réelle, pose une contrainte importante : la distance de sélection des objets est limitée par les capacités d'extension d'un bras humain. Une solution à ce problème a

[18]En référence au bras téléscopique de l'Inspecteur Gadget, un héros de dessin animé.

donc consisté à allonger la longueur du bras virtuel en modifiant le ratio d'extension par rapport au bras réel.

Ainsi, le bras virtuel s'allonge de façon non linéaire par rapport au bras réel. En-dessous d'une certaine distance *corps - bras de l'utilisateur* déterminée par une valeur seuil, l'allongement du bras virtuel est linéaire par rapport au bras réel. Au-dessus, la longueur suit une fonction non linéaire. R_v est la longueur du bras virtuel, R_r la longueur du bras réel, D une distance seuil et k un coefficient multiplicateur. On calcule R_v en utilisant une fonction de mapping non linéaire :

$$R_v = \begin{cases} R_r & \text{si } R_r < D \\ R_r + k * (R_r - D)^2 & \text{sinon} \end{cases} \tag{2.2}$$

Cette technique permet, comme dans le monde réel, de tendre le bras pour attraper un objet, bien que le rayon d'action soit plus grand. En outre, elle s'utilise aussi bien pour les distances faibles qu'élevées. Enfin, précisons que la manipulation est centrée sur la main virtuelle (par exemple une rotation d'un objet), ce qui est un atout par rapport aux techniques de lancé de rayons. Song *et al.* [SN92] ont également imaginé une technique similaire basée sur l'association d'une fonction non-linéaire au mouvement de la main réelle. D'après leur expérience, l'utilisateur doit être capable d'atteindre n'importe quelle position dans un vaste volume de données.

Le principal inconvénient de cette technique est qu'elle ne permet pas d'atteindre des objets qui sont vraiment trop lointains dans la scène, à moins de prendre un coefficient k élevé, ce qui ne s'avère pas très pratique, puisque le bras s'allonge trop rapidement. La figure 2.17 montre le déroulement d'une sélection GoGo. Nous y trouvons une jauge munie de curseurs qui permettent de connaître R_r par rapport à D. À tout moment, l'utilisateur peut savoir s'il est en mapping linéaire ou pas. Sur la figure, la partie verte à jaune (partie basse de la jauge) est celle où $R_r < D$ et jaune à rouge (partie haute de la jauge) celle où $R_r \geq D$.

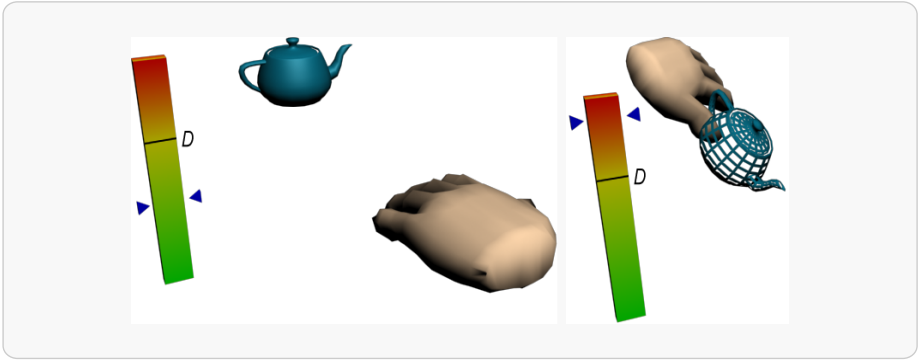


Fig. 2.17: Sélection par GoGo.

Bowmann *et al.* [BH97] ont défini trois variantes au GoGo de Poupyrev *et al.* [PBWI96]. D'une part, le fast GoGo : ici le bras virtuel utilise une fonction

de mapping non linéaire par rapport au bras réel sans valeur seuil. De fait, le bras virtuel s'allonge très rapidement, d'où le nom de fast GoGo. La fonction de mapping est la suivante :

$$R_v = R_r + k * R_r^2 \tag{2.3}$$

En regard de la fonction de mapping utilisée (voir ci-après), le fast GoGo permet d'atteindre rapidement les objets, mais la sélection est moins précise qu'avec le GoGo classique.

Avec le stretch GoGo, trois régions concentriques de l'espace sont définies autour de l'utilisateur. Lorsque la main physique se situe dans la région intermédiaire, la longueur du bras est constante. Si le bras physique est dans la région éloignée du corps, dans la région externe, le bras virtuel grandit à un taux constant. De manière similaire, avec le bras physique dans la région la plus proche du corps, la longueur du bras est réduite à un taux constant. L'utilisateur peut ainsi attraper n'importe quel objet dans la scène, quelle que soit la distance à laquelle il se trouve. Une jauge, affichée dans la scène, indique à l'utilisateur la distance *corps - main* dans les trois zones. Nous avons constaté que cette technique manque de précision, notamment pour arrêter la main virtuelle sur un objet, car il est difficile de repositionner rapidement la main dans la région intermédiaire. La fonction de mapping est la suivante (l représente une distance en centimètres ou en mètres) :

$$R_v = \begin{cases} R_r - l & \text{si } R_r < D_1 \\ R_r & \text{si } D_1 < R_r < D_2 \\ R_r + l & \text{si } R_r > D_2 \end{cases} \tag{2.4}$$

La figure 2.18 montre les trois zones utilisées avec la métaphore stretch GoGo. La zone « + » permet de faire avancer l'avatar de la main, tandis que la zone « − » permet de le faire reculer. Enfin, la technique de l'indirect gogo repose sur l'utilisation d'une souris $3D$. La main virtuelle est déplacée à l'aide des boutons de la souris. Le bouton de droite éloigne la main virtuelle, tandis que le bouton de gauche la rapproche. Le bouton du milieu sert à sélectionner les objets de la scène virtuelle. Il s'agit de la métaphore GoGo la plus précise, mais aussi la plus lente à utiliser.

2.4.3 Sélection dirigée

Définition Une sélection dirigée consiste, du point de vue de l'utilisateur, à indiquer dans quelle direction se situe l'objet à sélectionner.

En sélection dirigée, l'utilisateur indique dans quelle direction se situe l'objet qu'il souhaite manipuler. Il peut le faire en le pointant du doigt, en le fixant du regard, en utilisant conjointement ses deux mains, en précisant à quelle position

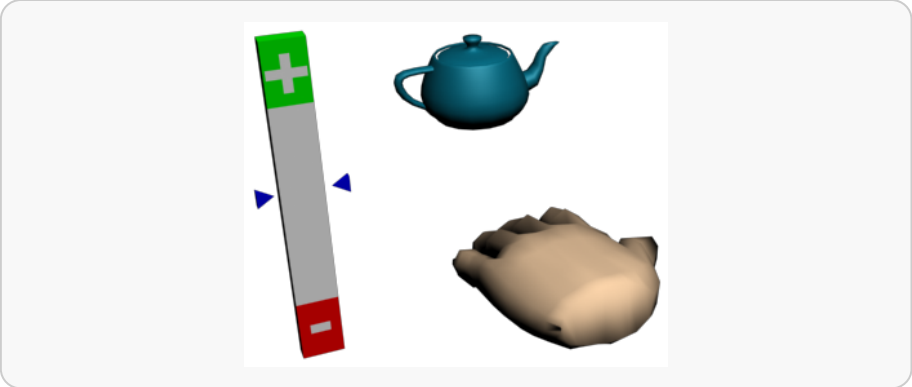


Fig. 2.18: Sélection par stretch GoGo.

il se trouve ou encore en donnant une position relative par rapport à un autre objet.

Direction pointée du doigt

L'utilisateur pointe du doigt[19] vers l'objet qu'il désire sélectionner, ce qui est très familier. Cette métaphore de sélection nécessite que l'index soit suivi dans l'espace, ce qui suppose un capteur de position attaché au doigt. Cette technique n'est pas sans rappeler la sélection par lancer de rayon vue plus haut. Elle souffre de limitations similaires comme par exemple la difficulté à sélectionner un objet trop petit ou trop lointain.

Direction du regard

La métaphore de sélection par le regard[20] est très intuitive, car l'utilisateur regarde ce qu'il souhaite sélectionner (figure 2.19 gauche), mais nécessite que la tête soit repérable dans l'espace. Tanriverdi *et al.* [TJ00] ont présenté une métaphore d'interaction basée sur le suivi des mouvements des yeux pour sélectionner des objets. Une caméra suit le mouvement de l'œil. Cette métaphore se révèle plus naturelle à utiliser que les systèmes de pointage tenu en main – comme par exemple le Wand, bien mois fatigante et contraignante, plus précise et plus rapide. Pour confirmer la sélection, il suffit de fixer du regard l'objet désigné pendant un court laps de temps, et pour le désélectionner de regarder ailleurs.

Néanmoins, cette métaphore force l'utilisateur à regarder dans une direction précise ; on ne peut donc pas sélectionner un objet avec le regard et, au même instant, réaliser une autre tâche qui nécessite la vision.

Direction du regard et de l'index

Un rayon virtuel partant de la tête et passant par l'index de la main de l'utilisateur permet de sélectionner un objet (figure 2.19 de droite). Cette métaphore

[19] a. *pointer-directed*
[20] a. *gaze-directed*

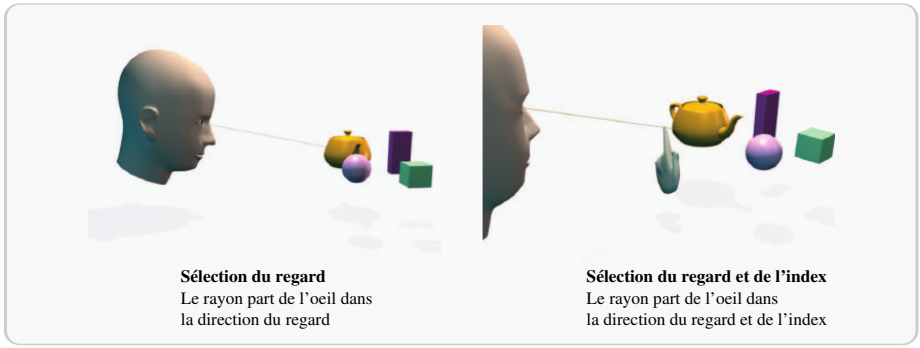


Fig. 2.19: Sélection dirigée du regard (à gauche) et du regard et l'index (à droite).

nécessite de suivre à la fois la tête et l'index, qui sont utilisés conjointement. Pierce *et al.* [PFC+97] la nomment sticky finger, mais on peut également la trouver sous le terme crosshair-directed.

Les auteurs définissent en outre plusieurs formes dérivées. Elles sont dites Image Plane, car l'utilisateur interagit avec les projections bidimensionnelles des objets tridimensionnels de la scène sur son plan image. Cela fonctionne de la même manière que sur un écran d'ordinateur classique (l'écran constitue alors le plan image), quand on déplace le pointeur de la souris au dessus d'un objet $3D$. Ici le pointeur de la souris est remplacé par le doigt, que l'utilisateur bouge devant lui, bras tendu. Bowman affirme justement qu'on peut considérer qu'il s'agit simplement d'une méthode de Ray-Casting où un rayon virtuel émane du point de vue de l'utilisateur et passe par sa main.

Le head crusher diffère du sticky finger, en ce sens que l'utilisateur emploie à la fois son pouce et son index. Le système détermine quel objet se situe entre les doigts de l'utilisateur, en lançant un rayon de sélection dans la scène, de l'œil dominant à l'objet et passant par le point médian entre l'index et le pouce. La figure 2.20 montre le fonctionnement du head crusher.

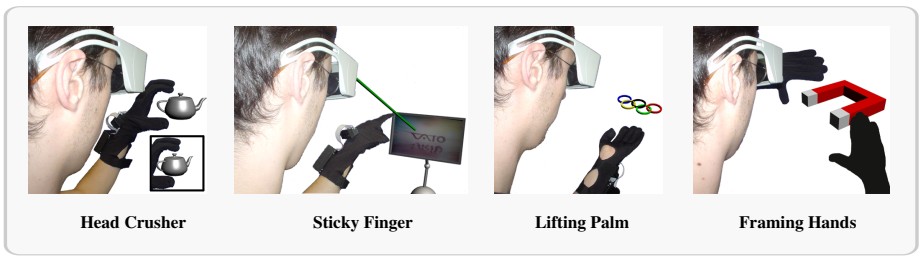


Fig. 2.20: Sélection par Head Crusher et ses dérivées.

La technique du lifting palm permet à l'utilisateur de sélectionner l'objet qui se trouve au creux de sa main. L'utilisateur place sa paume ouverte devant lui. Un rayon invisible partant de son œil dominant et passant par sa main permet

de sélectionner l'objet intersecté. La figure 2.20 permet de mieux comprendre le fonctionnement du lifting palm.

La dernière technique, le framing hands, utilise les deux mains simultanément pour sélectionner les objets. L'utilisateur positionne ses mains de manière à ce qu'elles forment un cadre qu'il peut agrandir ou réduire à volonté. Le système lance un rayon invisible passant par l'œil dominant et le point médian vers les objets. La figure 2.20 montre le fonctionnement du framing hands.

Schmalstieg *et al.* [SEaS99] matérialisent le plan à l'aide d'une tablette transparente en plexiglas que l'utilisateur tient dans sa main non dominante. Il est possible de sélectionner un ou plusieurs objets de deux manières. D'une part, l'utilisateur entoure à l'aide d'un stylo un ou plusieurs objets qui sont visibles par transparence sur la tablette. D'autre part, l'utilisateur peut centrer un objet au milieu de la tablette pour le sélectionner.

Les techniques précédentes d'Image Plane fonctionnent bien, et reprennent des principes utilisés dans le milieu cinématographique et artistique, mais elles posent cependant quelques problèmes. D'une part, une fatigue au niveau des bras peut survenir, puisque l'utilisateur est obligé de les tendre vers l'objet. D'autre part, des phénomènes d'occlusion peuvent apparaître lorsque le bras ou la main se trouve devant l'objet à sélectionner, ou devant une partie de la scène.

Direction du torse

Bowman *et al.* [BKH98] utilisent le corps de l'utilisateur, plus précisément son torse[21] pour indiquer dans quelle direction se déplacer et sélectionner un objet (figure 2.21). Mais cette technique s'avère peu pratique lorsqu'il s'agit de sélectionner un objet, puisque l'utilisateur doit éventuellement se pencher ; de plus, cette technique nécessite un capteur de position et d'orientation pour le torse. Par contre, elle permet de faire d'autres activités qui utilisent le reste du corps, mais pas le torse (regarder ailleurs, manipuler un objet, ...).

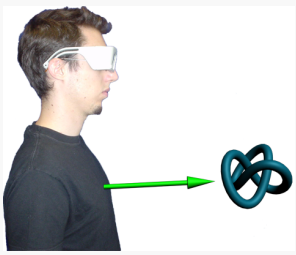


Fig. 2.21: Sélection dans la direction du torse.

En utilisation courante, cette métaphore ne donne pas de très bons résultats : elle manque de précision, et l'utilisateur doit se contortionner dans tous les sens lorsque l'objet n'est pas à la même hauteur que son torse.

[21] a. *torso-directed*

Direction par contrôle physique

L'utilisateur tient dans une main un joystick[22] par exemple, et de l'autre il le manipule pour orienter dans l'espace un rayon laser virtuel qui part du joystick et qui pointe vers l'objet à sélectionner. La figure 2.22 permet de mieux comprendre comment fonctionne cette métaphore.

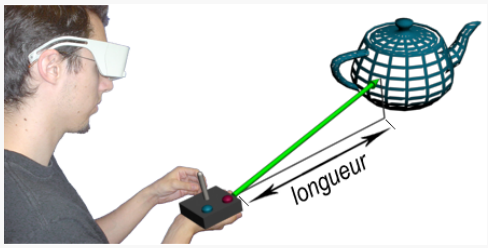


Fig. 2.22: Sélection par contrôle physique.

Manipuler un joystick n'est pas très efficace en $3D$, car le contrôle du déplacement du curseur est trop peu précis. En effet, il faut à la fois pouvoir le manipuler doucement lorsqu'il est proche de l'objet à sélectionner, et rapidement lorsque la cible est loin dans la scène. Or, l'amplitude d'un joystick n'est pas vraiment suffisante pour y associer une fonction de correspondance non-linéaire, comme on le fait pour la technique *GoGo* vue plus haut.

Direction par coordonnées

L'utilisateur indique au système dans quelle direction[23] le rayon de sélection doit pointer (par ex. « tourne à gauche de 45 degrés »). Ces informations sont passées au système par un mécanisme annexe (reconnaissance vocale, clavier, ...). La figure 2.23 montre le principe de fonctionnement.

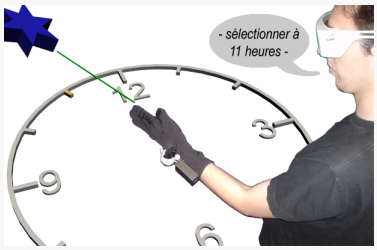


Fig. 2.23: Sélection par coordonnées.

[22] a. *device-directed*
[23] a. *coordinate-directed*

Direction relative

La direction peut être spécifiée relativement à un ou plusieurs objets de la scène virtuelle[24] (par ex. « à gauche de la voiture »). L'utilisateur indique au système ces informations par un système de reconnaissance vocale ou un clavier par exemple.

Le problème majeur de cette métaphore est que lorsque l'utilisateur donne une direction relative, plusieurs objets peuvent entrer en compétition pour être sélectionnés. Cette technique ne peut donc s'appliquer qu'à des cas bien particuliers où les objets de la scène ne sont pas trop proches les uns des autres.

2.4.4 Sélection par commande vocale

Définition Lors d'une sélection par commande vocale l'utilisateur emploie un mécanisme de reconnaissance vocale pour désigner et sélectionner explicitement un objet.

L'utilisateur nomme explicitement (par la voix) quel objet il désire sélectionner, ce qui nécessite bien entendu un mécanisme de reconnaissance vocale. La dénomination des objets peut se révéler ambiguë en pratique, surtout pour des objets qui portent le même nom (exemple : plusieurs sphères rouges dans la même scène). De plus, l'utilisateur a besoin de connaître le nom de chaque objet de la scène.

2.4.5 Sélection dans une liste de choix

Définition La sélection dans une liste de choix consiste à désigner un ou plusieurs objets parmi un ensemble de choix possibles, regroupés au sein d'une liste.

En sélectionnant un objet dans une liste, l'utilisateur est certain de ne pas manquer sa cible, pour peu qu'il sache à quelle entrée du menu correpond tel objet de la scène. Nous allons voir essentiellement deux techniques de menus : les menus issus de la $2D$ et les menus $3D$. Les menus seront étudiés plus en détail dans la section consacrée au contrôle du système (section 2.6).

2.5 La manipulation

Une des plus importantes formes d'interaction est la spécification de la position d'un objet et/ou son orientation dans le monde virtuel. Cette interaction peut soit être réaliste, c'est-à-dire que l'utilisateur prend et bouge l'objet virtuel exactement comme il le ferait dans le monde réel, soit être irréaliste, lorsque l'utilisateur bouge l'objet d'une manière qui n'a pas d'analogue dans le monde physique. Mine [Min95] indique que pour une tâche de manipulation, trois paramètres doivent être spécifiés lorsque l'on manipule un objet :

- son changement de position ;
- son changement d'orientation ;

[24] a. *landmark-directed*

– son centre de rotation.

Avant qu'un objet puisse être manipulé, il doit avoir été sélectionné. C'est pourquoi la manipulation est souvent fortement liée à sélection. La sélection est définie comme la désignation, puis la validation de la sélection d'un objet. La manipulation consiste à positionner et à orienter un objet sélectionné dans l'espace. Dans la pratique, sélection et manipulation sont souvent liées, sans que l'utilisateur ait explicitement besoin de passer de l'une à l'autre.

Nous avons choisi de scinder la manipulation en 5 groupes de métaphores. Premièrement, les techniques de manipulation locales, qui permettent de manipuler les objets tenus dans la main. Deuxièmement, les métaphores distantes, lorsque les objets sont hors de portée. Troisièmement, la manipulation bi-manuelle qui repose sur l'utilisation conjointe des deux mains. Quatrièmement, la manipulation contrainte quand le nombre de degrés de liberté est volontairement restreint. Enfin, cinquièmement, nous verrons le groupe des métaphores de manipulation hybrides, qui reposent sur l'utilisation de plusieurs modalités combinées : voix, gestes, etc.

2.5.1 Manipulation locale

Définition Une manipulation locale, ou centrée sur l'objet[25], est une opération qui se comporte comme si le centre de manipulation était celui de l'objet tenu en main.

Dans la vie de tous les jours, nous manipulons les objets localement, c'est-à-dire que nous les tournons et les déplaçons par rapport au centre de rotation du poignet. C'est la main qui agrippe l'objet et les rotations ou translations successives se feront autour du poignet. On parle alors de manipulation directe lorsque c'est le corps de l'utilisateur qui permet de manipuler localement l'objet d'intérêt. Nous verrons qu'elle peut être également indirecte lorsqu'un avatar de la main de l'utilisateur vient se porter à hauteur de l'objet, ou inversement qu'une copie de l'objet se rapproche de l'utilisateur.

Manipulation directe et indirecte

Comme dans la réalité quotidienne, il est possible d'utiliser directement les mains physiques pour manipuler un objet. L'utilisateur porte des gants munis de capteurs de position et d'orientation et ses gestes sont suivis dans l'espace pour déterminer quand débute et se termine l'interaction. L'avantage d'une telle technique est de diminuer les intermédiaires matériels entre l'objet virtuel et l'utilisateur, encore faut-il que les deux soient suffisamment proches. L'inconvénient le plus notable est que la main réelle vient cacher cet objet ; on ne sait plus très bien s'il est attrapé, en cours de déplacement, ou relâché.

Métaphore HOMER

Bowman *et al.* [BH97] ont inventé une technique hybride entre le lancer de rayon et les techniques de *GoGo* permettant de combiner les avantages de ces

[25] a. *object centered*

deux techniques. Il s'agit du *HOMER*, pour *Hand-centered Object Manipulation Extending Ray-casting*, dont nous allons détailler le fonctionnement. L'utilisateur active le rayon virtuel qui pointe sur l'objet désiré. En appuyant sur un bouton du périphérique d'interaction, la main virtuelle se déplace jusqu'au centre de l'objet sélectionné. Ce dernier peut alors facilement être manipulé – notamment pour les rotations dans l'espace, directement avec la main.

Quant la manipulation est terminée, la main virtuelle retourne à sa position d'origine, c'est-à-dire à l'endroit où se trouve la main réelle. La figure 2.24 montre le principe de fonctionnement suivant les différentes étapes définies par Bowman. Le rayon laser est d'abord affiché en rouge, puisque l'objet ne se trouve pas sur sa trajectoire. Puis, lorsqu'il y a intersection, le rayon devient vert. Enfin, l'avatar de la main de l'utilisateur se rapproche jusqu'à l'objet et la manipulation commence.

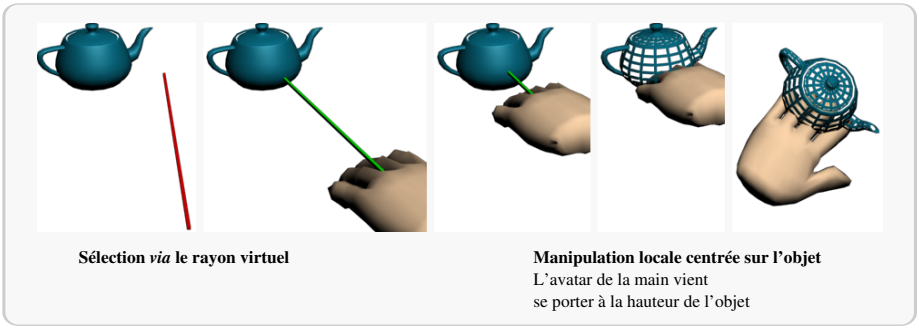


Fig. 2.24: Manipulation par HOMER.

Les auteurs ont imaginé deux versions du HOMER : direct HOMER et indirect HOMER. Dans le cas du direct HOMER, lorsque l'utilisateur saisit l'objet pour le manipuler, le fait d'éloigner sa main réelle entraîne un éloignement identique de l'objet et de l'avatar de la main. La distance maximum d'éloignement et de rapprochement de l'objet et de l'avatar est contrainte par la longueur du bras de l'utilisateur. Avec l'indirect HOMER un contrôle plus fin peut être réalisé en utilisant deux boutons d'une souris $3D$ (un bouton pour éloigner l'objet et l'avatar, et un bouton pour les rapprocher). Dans ce cas, la distance de déplacement n'est pas limitée.

2.5.2 Manipulation distante

Définition Une manipulation est distante lorsque le centre de manipulation n'est pas celui de l'objet manipulé.

En réalité virtuelle, il est assez fatigant de se déplacer à portée de l'objet pour le manipuler localement. En effet, avant chaque manipulation il va falloir se déplacer dans la scène avant de pouvoir modifier localement sa position et son orientation. Pour manipuler un objet distant sans se déplacer, on le fait le plus fréquemment *via* un rayon virtuel sur lequel vient s'accrocher l'objet d'intérêt. Plus la distance entre l'origine du rayon et le point d'accroche sur l'objet

est grande, plus les déplacements de l'objet seront importants. Inversement, il sera délicat de faire un déplacement précis lorsque l'objet est loin. En effet, les mouvements parasites de la main de l'utilisateur, c'est-à-dire ses tremblements, amplifient les mouvements involontaires de l'objet selon la longueur du rayon.

Ainsi, le déplacement est rapide, mais la rotation est délicate, surtout quand elle doit être centrée sur l'objet. Il faut constamment tourner légèrement l'objet, puis le relâcher, puis recommencer la rotation avant de pouvoir l'observer sous un autre angle. Puisque le centre de rotation n'est pas celui de l'objet, celui-ci pourra se retrouver hors du champ visuel de l'utilisateur, voire dans son dos.

Toutes les autres techniques de rayon, comme le rayon à ouverture, le rayon dirigé du regard, etc. souffrent du même problème quant à la rotation distante. La technique HOMER, citée au paragraphe précédent, permet cependant de sélectionner l'objet avec un rayon laser, et de le manipuler localement, puisque l'avatar de la main de l'utilisateur vient se porter à sa hauteur. Le cas du rayon dirigé par le regard est assez particulier, parce qu'il est délicat de l'employer pour la manipulation, les yeux étant constamment fixés sur l'objet.

2.5.3 Manipulation bi-manuelle

Définition Une manipulation est dite bi-manuelle lorsque l'on utilise conjointement nos deux mains pour manipuler un objet virtuel.

L'utilisation conjointe des deux mains permet de réaliser des tâches complexes. Encore peu développé en réalité virtuelle, ce type d'interaction reste difficile à mettre en œuvre. En effet, elle nécessite un matériel de précision (des gants de données) qui n'entrave pas les sensations de l'utilisateur. La manipulation bi-manuelle repose notamment sur la notion de chaîne cinématique qu'il est nécessaire de bien comprendre avant de pouvoir concevoir des outils de manipulation adaptés à certaines tâches virtuelles.

Modèle de la chaîne cinématique

La chaîne cinématique représente toute structure composée de liens rigides ou semi-rigides reliés entre eux. Chez l'être humain, le bras forme une chaîne cinématique équivalente à un système hiérarchique dont l'épaule est le repère de référence. En effet, lorsque l'épaule bouge, tout le bras – c'est-à-dire tout le reste de la chaîne – bouge. À l'inverse, lorsque le poignet bouge, seule la main bouge. Ainsi, un bras peut être considéré comme une hiérarchie à plusieurs niveaux de repères de référence.

Les joints font le lien entre les différentes parties rigides de la chaîne cinématique. Plus le joint est proximal, c'est-à-dire haut dans la hiérarchie, plus les mouvements seront amples et peu précis, et plus le joint est distal, plus le mouvement sera d'amplitude faible et précis. On parle de granularité.

Plus le joint est proximal, plus la granularité est grosse, et sa contribution lors du mouvement de la chaîne dure également plus longtemps. La main est l'extrémité distale libre et l'épaule est l'extrémité proximale.

Système Main dominante / Main non dominante

Le principe de la chaîne cinématique peut également s'appliquer à l'ensemble formée des deux bras. Dans ce cas, l'extrémité proximale est constituée de la main non dominante et l'extrémité distale par la main dominante. La main non dominante correspond à la main gauche (droite) chez le droitier (gaucher). On peut noter que lors d'un mouvement d'ensemble du bras, la main non dominante précède la main dominante dans les tâches bi-manuelles.

La main non dominante étant l'extrémité proximale de la chaîne cinématique formée par les deux bras, elle constitue donc un repère de référence pour la main dominante. Par exemple, pour enfoncer un clou dans une planche en bois, notre main non dominante tient le clou, le positionne et l'oriente, pendant que notre main dominante l'enfonce. Hinckley *et al.* [HPGK94] propose une application de visualisation de coupes du cerveau où l'utilisateur manipule une tête de poupée dans sa main non dominante pendant que la main dominante place le plan de la coupe (représenté par une plaque en plexiglas). Dans la majorité des actions bi-manuelles, les deux mains se coordonnent et coopèrent.

De fait l'usage des deux mains apporte une amélioration des performances. L'utilisation conjointe des deux mains permet de :

- **travailler en parallèle** : une synergie se crée permettant aux mains d'effectuer des tâches complexes ;
- **gagner du temps** : l'enchaînement de tâches séquentielles introduit des pertes de temps lors des transitions entre ces tâches ;
- **apporter plus d'informations** : le repère de référence, donné de la main non dominante, et la proprioception fournissent plusieurs informations permettant de réaliser plus rapidement et plus précisément des actions.

Une des premières utilisations faites en informatique concerne l'utilisation d'une tablette graphique et d'un stylo : la main non dominante tient la tablette alors que la main dominante utilise le stylo.

Mine *et al.* [MBJS97] décrivent une métaphore qui repose sur l'utilisation conjointe des deux mains pour voler dans une scène virtuelle. L'alignement des deux mains forme un vecteur qui donne la direction, sa longueur correspond à la vitesse de déplacement.

Le principe du repère de référence lié à la main non dominante est repris par Pierce *et al.* [PSP99]. Ils nomment cette technique les poupées vaudou, qui est d'ailleurs une variation du Monde en miniature de Stoakley *et al.* [SCP95]. Cette métaphore repose sur la manipulation de clones miniatures créés à la demande. Pour cela, l'utilisateur désigne l'objet d'intérêt à l'aide de la technique Head Crusher. Une miniature de l'objet et de son environnement proche est alors créée dans la main non dominante. Cette dernière sert de support à la manipulation, la main dominante étant dédiée au déplacement et à la rotation de la miniature. La figure 2.25 dépeint l'apsect bi-manuel de cette technique.

L'interaction HOMER Fishing Rod de Schmidt [Sch03] est une extension

Poupée vaudou
La main non dominante (ici la main gauche) tient la poupée, tandis que la main dominante la manipule.

Création du contexte de référence
La référence permet de manipuler plus précisément la poupée. Différents types sont disponibles : en haut à droite par cercle , en bas à droite par zone encadrante.

Fig. 2.25: Manipulation bi-manuelle par poupées vaudou.

bi-manuelle à la technique HOMER, c'est-à-dire qu'elle se fait à l'aide des deux mains. La main dominante effectue la sélection/manipulation par HOMER. La main non dominante permet de contrôler la distance de rapprochement entre l'objet et l'avatar de la main dominante. Cette distance est déterminée par l'ouverture de la main non dominante : lorsque la main est complètement ouverte, la distance est maximale, et lorsque le point est fermé, la distance est minimale (figure 2.26). On notera que dans cette métaphore, c'est l'objet qui se rapproche de l'avatar de la main et non l'inverse.

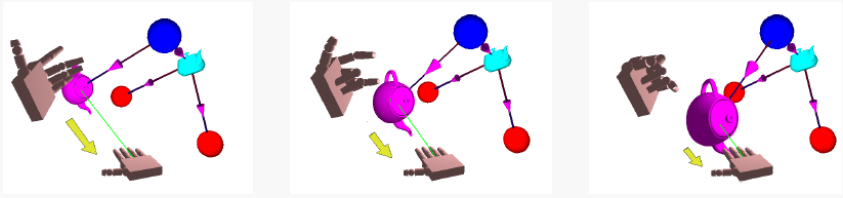


Fig. 2.26: Manipulation par HOMER Fishing Rod : la flexion du poing de la main non dominante détermine le rapprochement de l'objet vers la main dominante.

Kitamura *et al.* [KHK99] utilisent des baguettes chinoises pour une méthode de manipulation originale permettant d'attraper, déplacer et tourner un objet dans l'environnement virtuel (figure 2.27). Au choix, une des deux baguettes sert d'axe de rotation, tandis que l'autre donne l'amplitude de cette rotation. On peut également se servir conjointement des deux baguettes qui définissent alors un angle et un centre implicite.

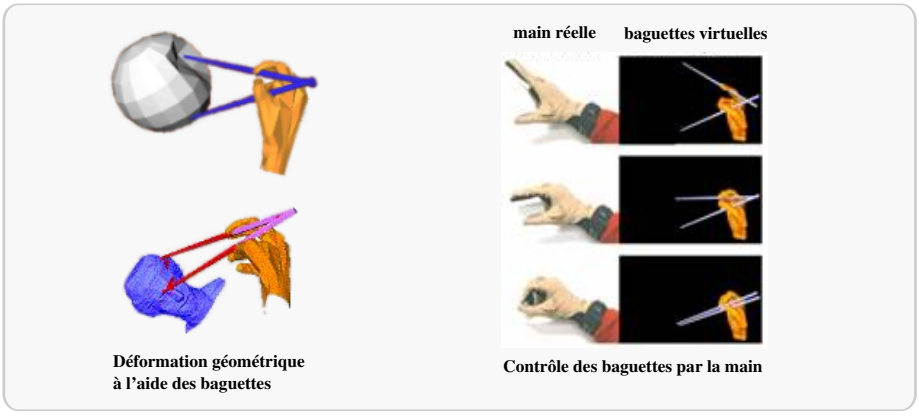


Fig. 2.27: Manipulation par baguettes chinoises.

2.5.4 Manipulation contrainte

Définition Une manipulation est dite contrainte lorsque le nombre de degrés de liberté associé à un objet virtuel est restreint de manière à faciliter sa manipulation.

Lorsque le nombre de degrés de liberté est trop important, certaines métaphores proposent de contraindre la manipulation afin d'éviter de surcharger cognitivement l'utilisateur. Par exemple, pour déplacer un cube dans un plan, seuls 2 degrés de liberté sont nécessaires. Or, les périphériques d'interaction autorisent jusqu'à 6 degrés de liberté. Il faudra donc restreindre les déplacements et rotations possibles de l'objet au strict nécessaire.

La manipulation, qu'il s'agisse de la rotation ou du positionnement, entraîne un certain nombre de difficultés. En effet, la manipulation d'un objet est généralement liée à l'environnement et/ou à d'autres objets de la scène. Il est donc impératif que la scène fournisse le maximum d'indices qui permettront, dans un premier temps, d'apprécier correctement l'orientation et la position d'un objet par rapport aux autres. Dans un deuxième temps, que l'environnement virtuel offre des indices de déplacement et d'orientation corrects et en nombre suffisant, afin que la manipulation se déroule dans les meilleures conditions possibles.

Les possibilités offertes par la réalité virtuelle en terme de manipulation ne sont pas forcément celles que l'on pourrait imaginer. Un des exemples les plus marquants reste le positionnement d'un objet dans l'espace. L'utilisateur éprouve plus de difficultés à placer précisément un objet lorsque celui-ci peut bouger librement dans le monde $3D$, que lorsque le mouvement est contraint sur un ou plusieurs plans ou axes. Aussi de nombreux auteurs ont décidé de contraindre certaines manipulations lorsque celles-ci doivent être précises [HTP+97, LST01]. Ainsi, on peut s'apercevoir que, tout comme dans le monde réel, toutes les situations ne se prêtent pas forcément à une manipulation $3D$. Il faut parfois contraindre la manipulation, c'est-à-dire diminuer le nombre de degrés de liberté.

Les widgets apportent des solutions de manipulation contrainte intéressantes. Conner *et al.* [BCSH+92] ont par exemple imaginé la sphère virtuelle[26] où le déplacement et la rotation d'un objet se font *via* l'intermédiaire d'un curseur de souris $2D$ qui se balade à la surface d'une sphère ; ceci permet de faire des rotations $3D$ avec une simple souris $2D$. Ils ont également imaginé d'utiliser des poignées[27] pour manipuler et aligner[28] plus facilement les objets de la scène. Par exemple, avec le concept de poignées, il est bien plus aisé de manipuler un objet compliqué, qui comporte beaucoup de faces orientées dans tous les sens, que les faces elles-mêmes.

Snibbe *et al.* [SHR+92] ont également utilisé des poignées accrochées aux objets pour pouvoir facilement réaliser une des trois opérations de Barr (fuselage, courbure et torsion). Ce système est avantageux pour plusieurs raisons. Le nombre de degrés de liberté est réduit à ceux disponibles *via* les poignées. Le contrôle des paramètres est explicite, c'est-à-dire que l'utilisateur sait directement à quoi correspond une poignée. Enfin, il est possible de combiner plusieurs poignées pour réaliser des déformations plus complexes.

Le système de contraintes sémantiques de Gösele et Stuerzlinger [GS99] permet de définir à l'avance quels sont les déplacements des objets qui sont autorisés, de manière à garantir que leur placement soit optimal dans la scène virtuelle. Smith *et al.* [SSS99] présentent un mécanisme très similaire. Kiyowaka *et al.* [KTY96] montrent qu'il est possible de manipuler des objets à plusieurs utilisateurs et/ou à deux mains. Pour cela, ils définissent un système de contraintes discrètes qui ne permet les déplacements et les rotations que selon certains axes et directions. Leur but est de proposer aux utilisateurs une manipulation la plus réelle possible, proche de leurs limitations physiologiques.

D'autres auteurs, comme Miller *et al.* [MZ99] imaginent des concepts comme les widgets haptiques, c'est-à-dire des widgets contraints par une force opposée aux mouvements de l'utilisateur. Ainsi, un cube posé sur une table ne la traversera pas visuellement, et lorsque l'utilisateur souhaitera le déplacer, son mouvement sera contraint dans le plan de sa surface de contact. Thalmann et Kallmann [TK99] proposent de la même manière un système entièrement contraint, de manière à ce que certaines actions seulement soient possibles.

Puisque la manipulation peut être contrainte, il est intéressant de disposer d'outils dont la forme rappelle leur fonction, c'est-à-dire leurs possibilités et limitations intrinsèques. Serra *et al.* [SPH+95] ont imaginé une panoplie d'outils, comme la fourchette pour déplacer des objets, le couteau pour faire des tranches, les ciseaux pour couper des lignes, etc. De manière similaire Koutek *et al.* [KP00] reprennent ce concept, en ajoutant certaines libertés physiques à l'outil. L'objet sélectionné est accroché au bout d'un ressort virtuel [KP01, KP02, KvHPB02], dont l'autre extrémité est tenue en main par l'utilisateur. Il devient ainsi possible de déformer une fourchette que l'utilisateur pique dans un atome lorsqu'il la tord, de pousser certains objets et de voir leur dureté à l'aide d'une sonde-ressort.

[26] a. *virtual sphere*
[27] a. *handlers*
[28] a. *snapping*

2.5.5 Manipulation hybride

Définition Une manipulation est dite hybride lorsqu'elle combine plusieurs modalités d'interaction (voix, mains, ...).

Les métaphores de manipulation peuvent combiner plusieurs modalités pour augmenter les possibilités d'interaction avec un environnement virtuel. Nous allons voir que l'utilisateur peut employer simultanément ses mains, sa voix, etc.

Dès le début des années 80, Bolt *et al.* [Bol80] ont imaginé d'utiliser conjointement la voix et les gestes pour manipuler des objets. Leur système *Put-That-There* permet de combiner les deux techniques d'interaction et de s'affranchir en partie des difficultés de l'autre. La reconnaissance vocale n'étant pas encore fiable, et la description des objets étant souvent une source de confusion, les gestes ont été utilisés pour sélectionner les objets. La manipulation se fait à l'aide des deux modalités. Par exemple, au lieu de dire « déplacer la voiture bleue sur la place de parking 114 », l'utilisateur pointera du doigt la voiture et la place de parking successivement en indiquant « déplacer ça ici ».

2.6 Le contrôle du système

Tout système informatique nécessite une interface entre l'homme et la machine, et la réalité virtuelle ne déroge pas à cette règle. Le contrôle du système repose sur un ensemble de techniques d'interaction dédié à l'émission de commandes dans le but de modifier l'état du système et ses paramètres, et le mode d'interaction qui y est associé. Il s'agit d'une tâche assez particulière, puisque c'est l'interface du système qui permet de contrôler ce même système. On entrevoit alors le rôle joué par les tâches de sélection et manipulation qui sont typiques à l'envoi, par exemple, d'une commande au système ou au choix d'une valeur dans une liste.

Aujourd'hui, dans le monde $2D$, il est impensable de travailler sur un ordinateur sans une interface graphique[29]. Un tel paradigme d'interaction repose la plupart du temps sur le système WIMP[30], constitué de fenêtres, d'icônes, de menus et d'un pointeur pour interagir sur ces entités graphiques. C'est en 1970 que Xerox propose un système basé sur le travail de psychologues s'intéressant au développement de l'enfant. Deux chercheurs, Jean Piaget et Jérôme Brunner imaginent un système graphique interactif simple :

- des icônes, que l'on peut reconnaître et comparer ;
- des symboles, qui permettent d'appréhender certaines notions abstraites ;
- une représentation graphique, à base d'objets graphiques déplaçables et orientables.

Comme dans le monde réel, une application de réalité virtuelle comprend des tâches de sélection, de manipulation et de navigation. Le contrôle du système est une tâche bien à part, qui sert de liant entre les autres tâches et l'application. Ce

[29] a. *Graphical User Interface* ou GUI
[30] Windows, icons, menus, pointer

n'est plus l'utilisateur qui décide comment effectuer telle ou telle opération, mais seulement quoi faire. Charge au système de s'occuper des détails du comment. Le rôle essentiel qui incombe au contrôle du système est donc de permettre à l'utilisateur d'entrer le plus efficacement possible des commandes. Aujourd'hui, le nombre de tâches est potentiellement très élevé, il importe donc de pouvoir regrouper et classer correctement les commandes. Les notions de conteneur, comme par exemple les fenêtres, et de hiérarchie, comme par exemple les menus, permettent de regrouper et classer correctement les commandes et faciliter leur accès.

Nous allons voir quels outils sont mis en œuvre pour passer des commandes au système, en détaillant autant que nécessaire les techniques que l'on rencontre en réalité virtuelle. La section 2.6.1, Outils graphiques, présentera l'évolution des interfaces graphiques utilisées, depuis la $2D$ jusqu'à la $3D$, en passant par la $2D^{\frac{1}{2}}$. Ensuite, nous discuterons des outils physiques en section 2.6.2 ; enfin, la section 2.6.3, Reconnaissance vocale et gestuelle, montrera comment les mains et la voix peuvent être employées pour activer une commande.

2.6.1 Outils graphiques

Les outils graphiques sont appelés des widgets[31]. Conner *et al.* [BCSH+92] définissent un widget comme étant un objet composé d'une géométrie et d'un comportement utilisés pour contrôler des objets graphiques ou une application informatique.

Les widgets sont tout simplement les menus, les boutons, les barres de défilement et autres composants graphiques qui permettent de faire le lien entre l'utilisateur et l'application. Nous allons voir dans un premier temps les outils graphiques utilisés sur les environnements $2D$, puis les tentatives d'importation de ces techniques dans un environnement $3D$, puis les outils $3D$ spécifiquement conçus pour la réalité virtuelle. Nous nous concentrerons sur le menu qui est le paradigme graphique le plus étudié. Nous verrons ensuite que le placement d'un tel paradigme dans l'espace $3D$ a une importance capitale pour son bon fonctionnement. Enfin, nous traiterons des outils graphiques dont l'interaction est contrainte.

Les interfaces $2D$

Définition Une interface $2D$ est une interface graphique que l'on trouve dans nos ordinateurs de bureau.

En tant qu'utilisateurs d'ordinateurs de bureau et de portables, notre communication avec la machine passe par une interface graphique $2D$, à laquelle nous sommes accoutumés. Cet type d'interface repose sur le principe du WIMP et son origine remonte au début des années 1980. Un tel environnement de travail, que l'on nomme aussi environnement fenêtré, repose sur plusieurs entités différentes. Premièrement, le bureau, qui représente l'espace de travail ; deuxièmement des fenêtres qui sont des conteneurs permettant de contrôler les

[31] pour *windowing gadgets*

programmes et manipuler les répertoires ; troisièmement des icônes qui représentent les fichiers répartis dans les répertoires (conteneurs). Plusieurs contrôles graphiques, ou widgets, servent à interagir avec l'utilisateur. L'ensemble de ces entités graphiques forment l'interface $2D$ de travail, qui se manipule à l'aide d'un curseur de sélection.

Selon la tâche que l'utilisateur souhaite réaliser, il choisira d'utiliser l'un ou l'autre de ces widgets. Au besoin, il faudra créer un nouveau composant, lorsque ceux qui existent déjà ne suffisent pas. Généralement, cela se fait en combinant plusieurs widgets de base au sein d'une entité plus complexe. Voici les commandes les plus utilisés, et les widgets qui y sont associées :

- **Lancer une commande** s'effectue à l'aide d'un bouton, d'un élément d'un menu ou encore d'une barre d'outil.
- **Activer ou désactiver une option** s'effectue à l'aide d'une case à cocher.
- **Saisir du texte ou une valeur numérique** s'effectue à l'aide d'un champ texte.
- **Sélectionner une valeur discrète dans un intervalle** s'effectue à l'aide d'un potentiomètre.
- **Choisir une option dans un jeu exclusif** s'effectue à l'aide de boutons radio.
- **Déplacer un objet** $3D$ s'effectue à l'aide d'un widget de translation.
- **Tourner un objet** $3D$ s'effectue à l'aide d'un widget de rotation.

La figure 2.28 montre les widgets $2D$ les plus courants : boutons, case à cocher, cases à choix exclusif, champ texte, valuateur horizontal, outil de translation et de rotation, etc.

Les interfaces $2D^{\frac{1}{2}}$

Définition On appelle interface $2D^{\frac{1}{2}}$, toute interface $2D$ importée dans un environnement $3D$.

Disposant d'un type d'interface d'un nouveau genre, mais ne disposant pas d'outils de contrôle du système, les chercheurs ont d'abord réflechi à inclure dans les environnements virtuels les outils qui fonctionnaient bien sur les environnements de bureau. Une des toutes premières tentatives vient du travail de Butterworth *et al.* [BAHO92] ; ils partent des barres d'outils $2D$ et proposent une barre suspendue devant l'utilisateur sur laquelle sont disposés des boutons ayant une épaisseur visuelle. Dans le même temps, Brookshire *et al.* définissent un widget comme étant un objet géométrique qui possède un comportement. Partant de là, ils proposent toute une série de widgets dans l'esprit des interfaces $2D$ mais étendues aux environnements $3D$. Ils imaginent par exemple la sphère virtuelle[32] qui contraint le déplacement d'un pointeur de souris de bureau à la surface de la sphère, ce qui permet de faire des rotations $3D$ avec une simple souris $2D$. Ils proposent également de placer des poignées[33] sur les objets de la scène pour faciliter leur manipulation. Enfin, les auteurs ajoutent

[32] a. *virtual sphere*
[33] a. *handlers*

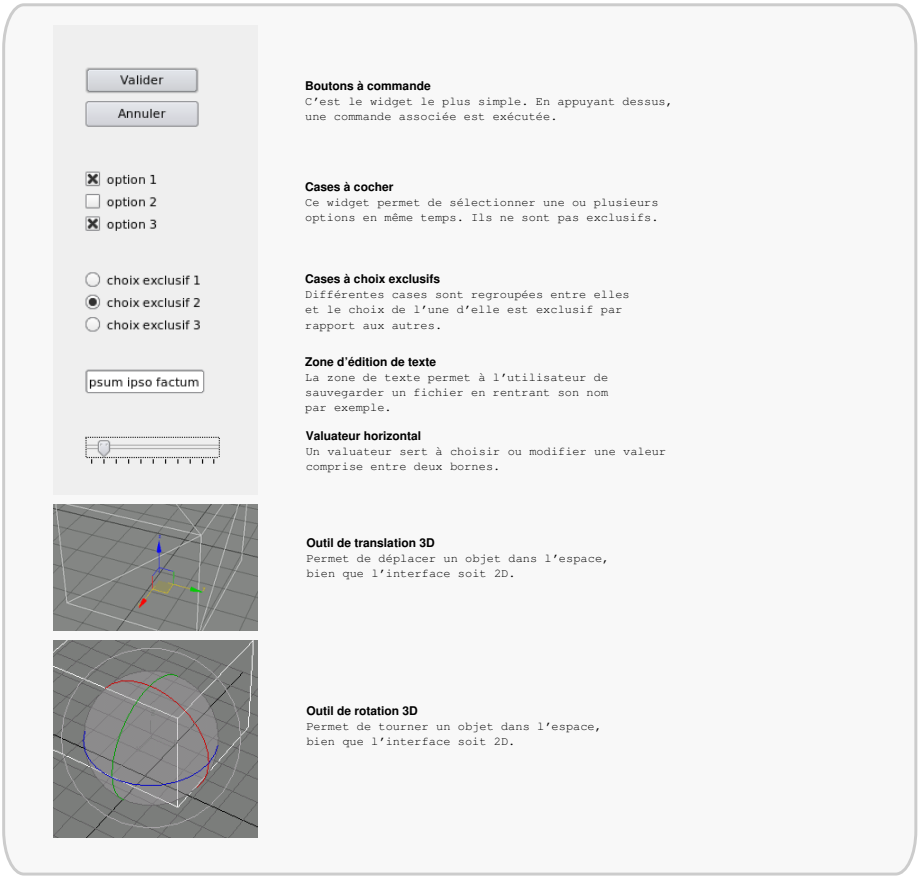


Fig. 2.28: Quelques widgets d'interface $2D$.

un widget d'alignement[34] des objets entre eux, ainsi qu'un autre pour changer leur couleur[35].

De manière analogue, Snibbe *et al.* [SHR+92] étudient l'association de poignées aux objets pour pouvoir facilement réaliser une des trois opérations de Barr [Bar84] (fuselage, courbure et torsion). Ce système est avantageux pour plusieurs raisons. Le nombre de degrés de liberté est réduit à ceux disponibles *via* les poignées. Le contrôle des paramètres est explicite, c'est-à-dire que l'utilisateur sait directement à quoi correspond une poignée. Enfin, il est possible de combiner plusieurs poignées pour réaliser des déformations plus complexes.

Un des premiers paradigmes graphiques utilisé en informatique pour le contrôle du système fut le menu linéaire. En effet, il paraissait naturel d'utiliser une liste linéaire pour sélectionner un document ou une action parmi un ensemble. Jacoby *et al.* [JE92] définissent un menu comme une collection de choix – généralement des options ou des commandes – qui est affichée à l'utilisateur. Dépendants de la puissance des ordinateurs de l'époque, et des technologies d'affichage limitées, les premiers menus étaient affichés en 2 dimensions, c'est-à-dire plats. Les auteurs choisissent de porter ces menus en $3D$ en les rendant positionnables et orientables dans l'espace, et les nomments menus $2D$ convertis, c'est-à-dire $2D^{\frac{1}{2}}$. Le pointeur de souris est remplacé par une interaction au laser virtuel qui part du bout de l'index de l'utilisateur. Il lui suffit alors de pointer l'élément souhaité.

Darken *et al.* [DL94] améliorent cette technique en rajoutant de la transparence aux menus, ainsi qu'un retour haptique lors de la manipulation de ceux-ci. Ainsi, il est possible d'importer de nombreux widgets $2D$ en réalité virtuelle, l'utilisateur a alors tout le loisir de les positionner et les tourner comme il le veut dans l'espace. De plus, la bibliothèque de widgets disponibles, et donc potentiellement injectables en $3D$, est énorme. Bien entendu, un certain nombre de problèmes apparaissent lors d'une telle opération de conversion. Les périphériques d'interaction sont différents ; alors que la souris de bureau est contrainte sur le plan de la table sur laquelle elle repose, en environnement immersif le périphérique bouge librement dans l'espace et la précision est bien moindre. Ainsi, le pointeur de souris qui était précédemment contraint dans le plan de la souris, est remplacé par un rayon virtuel qui est orientable dans l'espace. Il est donc facile de faire des erreurs de pointage, surtout lorsque l'élément à sélectionner est loin de l'origine du rayon. Agrandir la taille des composants constitue une solution envisageable, malheureusement elle se heurte à la taille limitée de l'environnement de réalité virtuelle. Un menu trop grand pourrait alors prendre trop de place.

Les interfaces $3D$

Définition On nomme interface $3D$ une interface graphique spécifiquement conçue pour un environnement $3D$.

Nous venons de voir qu'il est possible d'importer des widgets $2D$ dans un environnement de réalité virtuelle. Pourtant cette solution n'est pas toujours

[34] a. *snapping*
[35] a. *color picker*

adéquate, et certains chercheurs ont essayé d'imaginer des composants d'interfaces directement en $3D$. Les périphériques spécifiques à la réalité virtuelle et les degrés de liberté disponibles dans un environnement immersif permettent bien plus de possibilités d'interaction que celles que l'on trouve généralement en environnement de bureau. Il s'agit d'une discipline relativement jeune et seules quelques métaphores de contrôle du système existent actuellement, essentiellement des métaphores de menu. Nous allons donc nous intéresser essentiellement aux menus dans les paragraphes qui suivent.

Les menus circulaires Liang *et al.* [LG93] ont imaginé représenter la liste des choix du menu de manière circulaire, comme le montre la figure 2.29. Ce type de menu est spécifiquement destiné à une utilisation à l'aide de périphériques $3D$ tenus en main (gants de données, souris $3D$, ...). Bien que n'occupant que peu de place à l'écran un menu en anneau simple ne permet pas d'afficher un nombre important de choix. En effet, plus on place de choix sur la circonférence de l'anneau plus la sélection d'une entrée est difficile, puisqu'elle occupe de moins en moins de place.

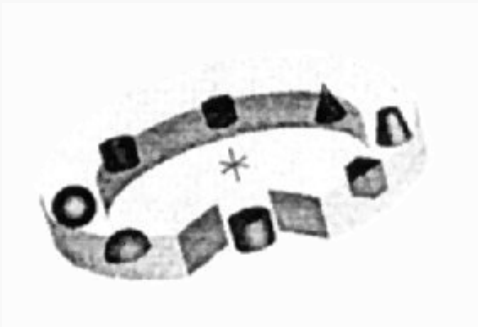


Fig. 2.29: Menu en anneau utilisé dans le modeleur JDCAD.

Comme de nombreux modeleurs, le système THRED – pour Two Handed Refining EDitor – de Shaw *et al.* [SG94] propose à l'utilisateur un nombre important de choix. Les auteurs ont imaginé de représenter la liste de ces choix circulairement, à l'instar de Liang *et al.* [LG93] et leur modeleur JDCAD. Ils l'ont dénommé le menu en anneau[36]. Pour sélectionner une entrée, l'utilisateur fait tourner le menu à l'aide d'un bat (bouton traqué à 3 états : enfoncé, relâché et à demi-enfoncé). Cette fois-ci, contrairement aux menus plats, ce n'est pas le curseur qui se déplace vers une entrée, mais l'entrée qui se déplace sous le curseur. L'anneau est affiché horizontalement, dans le plan XZ, et tourne autour de l'axe Y, en déplaçant latéralement le bat. Une petit curseur indicatif est affiché depuis le centre du menu en direction de l'utilisateur ; c'est donc toujours l'entrée qui est la plus proche de l'utilisateur qui est sélectionnée. Les auteurs ont proposé une version hiérarchique de ce menu, où les différents plateaux sont

[36] a. *ring menu*

empilés les uns au dessus des autres.

Wesche *et al.* [DW00] combinent la métaphore de sélection du lancer de rayon avec un menu en anneau hémisphérique. Le rayon est contraint dans un plan, de telle manière à ce qu'il reste toujours en contact avec une entrée du menu (figure 2.30).

Fig. 2.30: Menu en anneau hémisphérique à sélection par lancer de rayon contraint.

Plus récemment, Gerber *et al.* [GB04a] ont repris le concept de menu en anneau de Liang, mais en ne considérant cette fois-ci que l'hémisphère de l'anneau (figure 2.31) qui fait face à l'utilisateur. Le curseur de sélection reste fixe, tandis que le Spin Menu, comme l'ont nommé ses concepteurs, pivote suivant la rotation du poignet de l'utilisateur. Le menu est découpé en tranches de même largeur. Or, quand le curseur se situe à la frontière entre deux éléments de l'anneau lors de la validation, un mouvement, même imperceptible suffit à faire le mauvais choix. D'où l'idée d'un filtrage qui réduit sensiblement les parasites des mouvements de l'utilisateur lorsque celui-ci valide l'entrée souhaitée dans le menu.

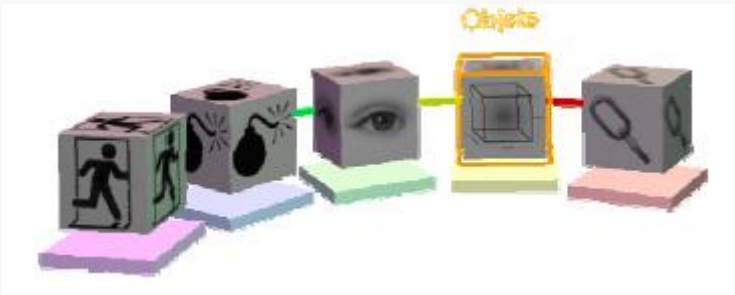


Fig. 2.31: Menu Spin, de forme hémisphérique.

Il faut également noter que les auteurs ont fait évoluer le système vers un menu hiérarchique [GB05] en empilant les hémisphères les uns sur les autres (figure 2.32). Le Spin Menu s'avère aussi bien indiqué pour les utilisateurs novices qu'expérimentés, puisque sa prise en main est très aisée.

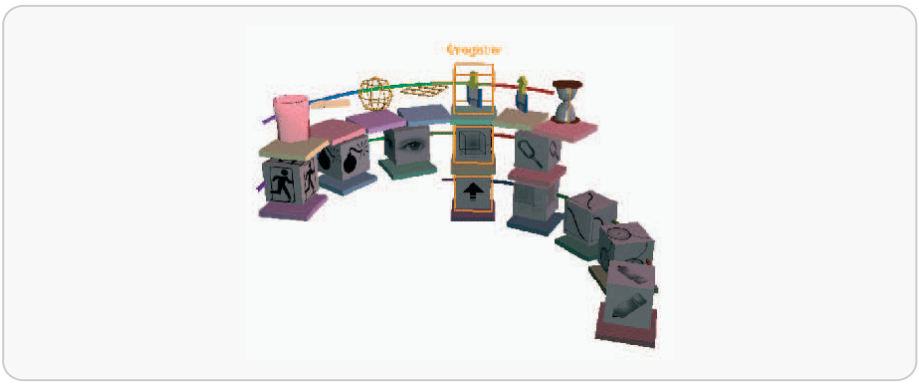


Fig. 2.32: Menu Spin hiérarchique.

Le menu *Command and Control Cube* Grosjean *et al.* [GC01] ont imaginé un menu tridimensionnel. Il s'agit du CCube ou C^3, pour Command and Control Cube. Les auteurs sont partis du concept des Marking Menus de Kurtenbach et Buxton [KB94], qui proposent de disposer les éléments du menu à équidistance d'un point central, comme des parts de tarte, pour en accélérer la sélection. Les auteurs du CCube ont proposer un mécanisme similaire aux raccourcis clavier[37], qui n'avaient pas d'équivalent dans les environnements immersifs.

Le CCube est un menu qui contient 27 cubes répartis en 3x3x3 boîtes (figure 2.33 de gauche). L'ensemble de ces cubes forme une structure cubique, celui du centre servant à la fois de point de départ à l'interaction sur le menu et d'opération d'annulation. Un pointeur sphérique de couleur clair est utilisé pour naviguer dans cette structure cubique (figure 2.33 de droite). L'utilisateur peut donc sélectionner une commande en faisant un unique geste dans la bonne direction, et ce quelle que soit son amplitude.

Pour les utilisateurs avertis, la manipulation du C^3 peut s'avérer très efficace ; en effet, puisque l'amplitude du mouvement n'entre pas en considération – il ne s'agit pas d'atteindre une position particulière, mais seulement de donner une direction –, la sélection d'un élément du menu peut être très rapide. Le fonctionnement est relativement simple, puisque l'utilisateur a seulement à appuyer sur un bouton de son périphérique, puis déplacer celui-ci dans la direction voulue, et enfin relâcher le bouton pour valider sa sélection. La sélection peut se faire en aveugle – sans affichage du C^3 – pour un utilisateur expert. Les auteurs proposent en outre de tenir compte de la direction du regard par rapport à la position du menu pour savoir s'il est nécessaire de l'afficher. Ce mécanisme agit

[37] a. *hotkeys* ou *shortkey*

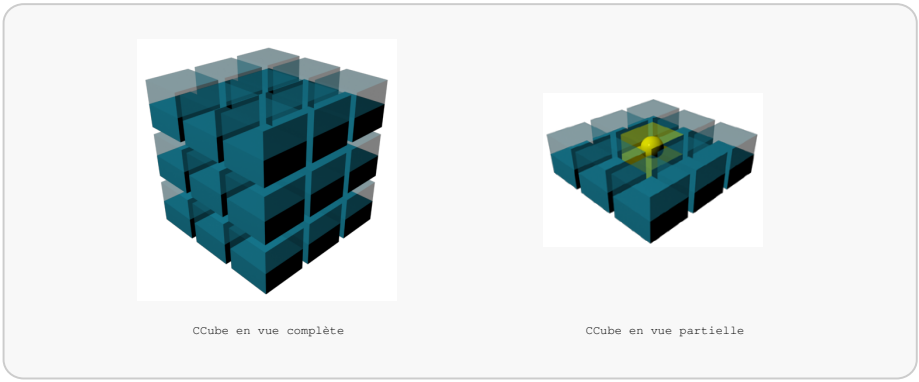


Fig. 2.33: Menu $3D\ C^3$ en vue complète (à gauche) et en utilisation normale (à droite).

donc comme un raccourci clavier. Les tests réalisés par les auteurs montrent que le C^3 est très rapide pour la sélection d'un élément du menu et qu'il occasionne très peu d'erreurs.

Le menu *TULIP* Bowman et Wingrave [BW01] ont conçu et réalisé un menu déroulant s'affichant dans le creux de la paume de l'utilisateur et sur le bout de ses doigts. Chaque élément de ce menu est affiché au bout d'un des doigts de l'utilisateur, ce dernier portant des gants de données Pinch Glove, sensibles au contact. La sélection est réalisée en pinçant le doigt correspondant à l'élément souhaité avec le pouce. La version simple et non hiérarchique de cette métaphore s'utilise à une main. À l'inverse, lorsque le nombre d'entrées est trop important, on utilisera au choix les deux mains, ou une version hiérarchique des menus qui sont dans ce cas affichés en cascade (figure 2.34).

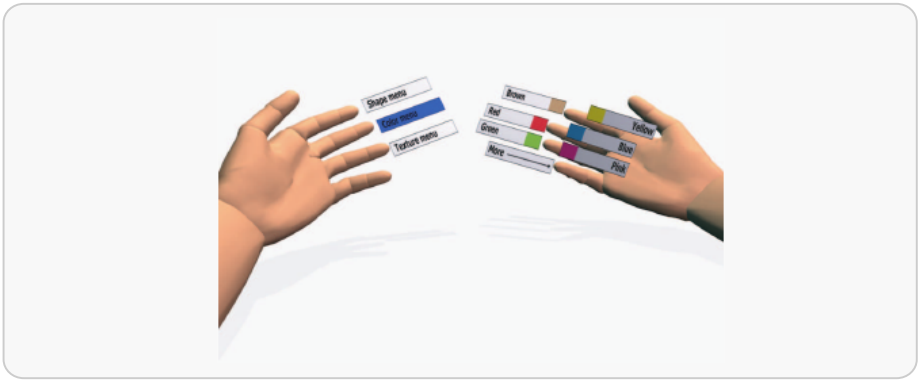


Fig. 2.34: Menu TULIP.

Cette technique est très bien adaptée à des menus comportant peu d'éléments ; en effet, l'utilisation du contact entre les doigts repose sur l'agilité des phalanges d'une main humaine, rendant la sélection à la fois précise et rapide.

À l'inverse, lorsque le nombre d'entrées est trop élevé, la métaphore rencontre quelques limitations majeures. D'une part, l'affichage des menus à proximité des mains peut engendrer des phénomènes d'occlusion et dans ce cas, la technique ne sera adaptée qu'à un environnement de réalité augmentée où le menu s'affichera toujours au-dessus des mains. D'autre part, l'affichage d'une entrée en face d'un doigt permet de bénéficier de l'aspect proprioceptif des mains : puisque celles-ci servent de référence, l'utilisateur n'a plus besoin de regarder le menu pour faire sa sélection. Lorsque le nombre d'éléments est trop grand, il faudrait décaler les entrées, ce qui nuit bien entendu à cet avantage. Comme pour le C^3 vu précédemment, la sélection peut se faire en aveugle, sans voir le menu.

Placement des menus dans l'espace $3D$

Le placement des menus joue un rôle très important, tant au niveau de leur lecture que de leur utilisation. Nous allons voir les differents types de positionnement possibles dans un environnement virtuel, selon trois objets de référence : le monde, le corps de l'utilisateur et un périphérique que l'utilisateur manipule.

Menu attaché au monde Les premiers menus à être utilisés en réalité virtuelle furent fixés au monde. Pour pouvoir bouger un tel menu, l'utilisateur doit d'abord l'attraper, par exemple avec un rayon laser virtuel, puis le déplacer et enfin l'utiliser. L'interaction avec un tel menu est assez fatigante, car il faut continuellement le manipuler, pour éviter l'occlusion du reste de la scène, ou de certains objets.

Menu lié au corps Mine *et al.* [MBJS97] suggèrent d'utiliser au maximum la proprioception pour améliorer les performances lors de tâches virtuelles. En particulier, pour les menus $2D$, ils proposent de placer un menu déroulant audessus de la tête de l'utilisateur, hors du champ de vision. Pour sélectionner un choix, il suffit de le baisser avec une main, à la manière d'un store vénitien. Il remonte ensuite en le lâchant. Les auteurs proposent de disposer les menus en fonction de leur utilité. Par exemple, en liant les fonctions du menu à celles du corps, il est judicieux de placer les menus qui concernent des choix visuels près de la tête de l'utilisateur. En outre, les entrées sont représentées par des icônes sans texte afin de minimiser la place qu'elles occupent.

Pierce *et al.* [PCvDR99] se sont intéressés à la place que prennent les barres d'outils et les menus dans un environnement virtuel. Pour eux, la disposition doit être centrée sur l'utilisateur pour que les barres entourent l'utilisateur, et n'obstruent que partiellement la vue de l'environnement virtuel. Les barres d'outils et les menus sont attachés au corps de l'utilisateur. Il peut donc se servir de sa proprioception pour atteindre n'importe quel élément d'une des barres d'outils qui l'entourent, sans même le regarder.

Menu lié à un équipement mobile Plusieurs auteurs ont imaginé qu'attacher les menus à des équipements physiques pourrait apporter une certaine facilité d'utilisation. Wloka *et al.* [WG95] ont proposé d'afficher un avatar virtuel d'un périphérique physique à 6 degrés de liberté dans l'environnement. Ainsi l'utilisateur dispose de la sensation tactile de l'outil (retour haptique passif, voir

définition à la sous-section 1.3.2 sur le toucher). À cet outil virtuel, le virtual tricorder, est attaché un menu. La sélection se fait à l'aide des boutons placés sur le dispositif d'interaction. Précisons que le menu est semi-transparent, ce qui évite les phénomènes d'occlusion des objets de la scène virtuelle. Le menu utilisé par les auteurs est plat, mais orientable dans les 3 dimensions de l'environnement.

Plusieurs auteurs ont essayé d'accrocher virtuellement des menus à des tablettes (interaction tablette et stylo) [BWHA98, LSH99, PTW98, SG97]. Cette manière de faire permet de contraindre l'interaction à un plan physique (le plan de la tablette), ce qui s'avère efficace pour la manipulation des menus et la sélection des entrées. En effet, d'une part la tablette fournit un retour haptique passif. D'autre part, le fait de contraindre l'interaction à un plan physique diminue les erreurs d'interaction, puisque le menu est plat ce qui correspond bien à un plan. La tablette est donc bien adaptée à l'interaction avec des menus plats.

La plupart du temps la tablette est utilisée avec un casque de réalité augmentée. Schmalstieg *et al.* [SEa98, SEaS99] ont utilisé une tablette transparente sur un équipement de type *Workbench*. L'image à afficher sur la tablette est re-calculée en permanence de manière à ce que l'utilisateur ait l'impression que les données sont affichées directement sur la tablette.

Outils graphiques contraints

Hormis les widgets "classiques", comme les boutons, les fenêtres ou les barres de défilement, il peut être intéressant de disposer d'outils soumis à certaines propriétés physiques, c'est-à-dire contraints. Miller *et al.* [MZ99] cherchent à rendre les outils d'interaction plus faciles d'utilisation. Pour cela, les auteurs ajoutent un retour haptique à leurs widgets, le but étant de fournir à l'utilisateur une information physique précisant ce qui vient d'avoir lieu sur tel ou tel objet graphique. Par exemple, en appuyant sur un bouton virtuel, il aura la sensation d'appuyer sur un bouton réel. En effet, une force s'opposera aux déplacements de sa main et simulera le comportement d'un bouton physique.

La manipulation des widgets $3D$ n'est pas toujours aisée, surtout lorsque l'utilisateur doit manipuler une partie du widget qui n'est que 1 ou $2D$ (barre de défilement par exemple) à l'aide d'un périphérique d'interaction qui a bien plus de degrés de liberté (un périphérique à 6 degrés de liberté par exemple). Pour diminuer ces effets d'imprécision qui pénalisent beaucoup la manipulation du widget, Lindeman *et al.* [LST01] ont imaginé de contraindre les widgets $3D$ sur des surfaces qu'ils nomment *surfaces simulées*. Par exemple, l'avatar de la main de l'utilisateur peut se déplacer librement dans l'espace, sauf que lorsqu'il se trouve au contact du widget barre défilante, il lui est impossible de passer au travers. De plus, des informations annexes permettent d'augmenter le retour (indices sonores pour la validation d'un choix, ombres portées de formes simples et illumination du bout de l'index virtuel lorsqu'il est en contact avec le widget). Cette méthode s'appelle le *clamping* et permet, lorsque l'on ne peut contraindre les mouvements de l'utilisateur sur une surface physique, d'améliorer les performances par rapport à une manipulation parfaitement libre. Les auteurs indiquent que les concepteurs d'interfaces devraient limiter le nombre de degrés de précision requis dans un système, et n'appliquer que les degrés nécessaires et suffisants, car certaines tâches ne peuvent concrètement se réaliser que si elles sont correctement contraintes.

2.6.2 Outils physiques

Dans de nombreuses applications $3D$, l'utilisation d'outils de la vie de tous les jours pour l'interaction peut permettre d'accroître les performances de l'utilisateur. Ces outils, que l'on nomme ici outils physiques, fournissent des moyens d'interaction directs, puisqu'ils sont issus du monde réel. Un des intérêts remarquables de ces dispositifs est d'offrir un support physique. L'utilisateur peut concrètement toucher les outils qu'il a entre les mains ; sa sensation d'immersion s'en trouve ainsi accrue.

Nous avons choisi de les regrouper en trois catégories distinctes. Premièrement, les dispositifs portables, comprenant essentiellement les assistants personnels et les téléphones portables que l'on utilise pour leur légèreté et leur taille réduite. Deuxièmement, les pointeurs laser que l'on utilise généralement pour montrer quelque chose sur un écran, et qui sont aussi utilisés pour remplacer un pointeur de souris. Enfin, les interfaces tangibles, qui sont des périphériques construits pour fournir un retour tactile de l'objet virtuel que l'on manipule, comme par exemple un menu.

Les techniques de cette sous-section pourraient être présentées comme des tâches d'interaction générale. Cependant, il nous semble pertinent de les placer ici parce qu'elles sont essentiellement utilisées pour contrôler le système.

Les dispositifs portables

Fitzmaurice [Fit93] propose d'interagir avec un environnement $3D$ à l'aide du prototype Chameleon. Ce dispositif, l'un des premiers du genre que l'on nomme aujourd'hui un assistant personnel[38], se présente sous la forme d'un écran qui tient dans la main. Des capteurs sont ajoutés sur les bords de l'appareil, pour le situer dans l'espace. Lorsque l'utilisateur le place au dessus d'une carte physique, il fournit un certain nombre de renseignements sur les régions survolées par l'appareil (météorologie, ...). Une seconde application concerne la visite de bibliothèques : il est possible d'avoir des renseignements sur le contenu des étagères et des livres, simplement en passant devant. On peut comparer le Chameleon à une loupe : il tient dans la main et dévoile des informations qui ne sont pas directement visibles à l'œil nu.

Watsen *et al.* [WDC99] utilisent un assistant personnel pour interagir avec l'environnement virtuel. L'idée est d'utiliser des widgets sur le périphérique pour contrôler des objets virtuels, ce qui permet de passer en douceur d'un environnement purement $2D$ (le PC de bureau), à un environnement $3D$ avec lequel il est souvent difficile d'interagir convenablement. D'une part, leur dispositif est utilisé pour contrôler le point de vue à l'aide du stylo, et d'autre part, l'utilisateur dispose de plusieurs widgets $2D$ pour changer la géométrie et la couleur de certains objets de la scène virtuelle.

Hartling *et al.* [HBCN02] proposent d'utiliser en combinaison Java $2D$ et CORBA pour réaliser une librairie de développement d'interfaces virtuelles nommée Tweek. Ainsi les utilisateurs interagissent par l'intermédiaire de l'interface

[38] a. *Personnal Assistant* ou *PDA*

graphique fournit par Java 2*D* et Java communique avec l'application de réalité virtuelle *via* CORBA. L'interface d'interaction est un ordinateur de poche de type PalmPilot.

L'interface développée par Brunelli *et al.* [BFB02] permet à plusieurs utilisateurs de travailler ensemble dans un même environnement virtuel. Chaque assistant personnel communique avec le serveur *via* les RMI de Java, et ces derniers permettent d'utiliser conjointement JAVA et CORBA et le protocole 802.11b à 11 Mb/s. Il est ainsi possible d'utiliser n'importe quel type de PDA supportant JAVA. L'interface utilisateur se compose d'une carte 2*D* active, avec laquelle l'utilisateur visualise sa position/orientation dans le monde virtuel et les régions qu'il peut atteindre directement, etc. Il est également possible de sélectionner des objets pour les connecter avec des données 2*D*, d'échanger des données multimédia avec les autres utilisateurs, ou de naviguer librement ou de manière guidée dans la scène.

Wagner *et al.* [WS03] ont également pensé utiliser un assistant personnel sur lequel est fixée une mini caméra, et avec laquelle ils réalisent un suivi (dans l'espace) de marqueurs positionnés dans une pièce. Pour cela, ils utilisent la librairie ARToolKit qui tourne directement sur l'assistant. La caméra enregistre la scène, puis l'ARToolKit reconnaît les marqueurs et ajoute des informations à la scène enregistrée, et enfin, la scène augmentée est affichée sur l'écran. Il n'y a donc pas de casque de réalité augmentée, ni de stéréovision, et le PDA peut être considéré comme une métaphore « see-through » qui permet de voir plus d'informations que celles qui sont disponibles dans le monde physique. Le dispositif peut éventuellement être secondé par un serveur qui traite les images *via* une communication sans fil. L'application de test est Signpost, qui affiche un certain nombre d'informations de navigation : plan auto-orienté, nom et fonctionnalité des pièces et des objets, etc.

Depuis quelques temps, de nouveaux types de portables font leur apparition : ils se situent à mi-chemin entre l'assistant personnel et l'ordinateur portable, et sont de la taille d'une feuille A4 ou un peu plus petits. Muller *et al.* [MCK03] utilisent un de ces WebPad pour la création d'environnements virtuels. Le graphe de scène correspondant à un environnement virtuel (c'est-à-dire la structure de la scène et des différents objets) est affiché sur ce WebPad en 2*D*, car cette interface facilite l'accès aux données textuelles et aux documents attachés aux objets de la scène. L'utilisateur dispose également d'une interface 2*D* classique, couplée aux librairies Open Performer et AVANGO, qui lui permet de concevoir rapidement un environnement virtuel complet.

Les pointeurs laser

Les pointeurs laser sont régulièrement employés dans les présentations et les conférences, car ils permettent de mettre rapidement en évidence certaines informations. Des auteurs ont détourné l'utilisation première de ces outils pour leur permettre d'interagir avec un environnement essentiellement 2*D*, mais également 3*D*. Davis *et al.* [DC00] présentent une métaphore d'interaction basée sur l'utilisation d'un stylo laser. Il est utilisé sur des écrans de grande taille (mur), derrière lesquels tout un ensemble de caméras analysent les spots des

lasers de plusieurs utilisateurs (interaction multi-utilisateurs). Chaque caméra observe une partie de l'écran, et l'ensemble des données des différentes parties est centralisé sur un serveur. Ce dernier détermine 3 types d'événements pour chaque spot : début du trait (chaque spot trace un trait sur l'écran), fin du trait, position courante du spot. Il est ainsi possible de dessiner à distance sur un écran par l'intermédiaire de stylos laser. Une reconnaissance des formes peut ensuite être envisagée pour lancer des actions ou simplement déplacer des icônes.

L'utilisation d'un pointeur laser offre des possibilités d'interaction assez intéressantes. Il est ainsi possible d'indiquer directement sans avoir à déplacer un pointeur de souris, le lieu/l'objet d'intérêt sur l'écran. Pour Wissen [Wis01], le principe de la détection des spots laser sur l'écran repose sur l'utilisation de caméras qui scrutent en permanence l'écran. Les coordonnées du spot sont ensuite calculées et transmises à un ordinateur graphique. Le stylo-laser comporte 3 boutons : 2 correspondent au clic-gauche et clic-droit, le 3ème permet de faire des cadres de sélection. Cette technique permet de simuler la souris de bureau.

Olsen *et al.* [OJN01] ont d'ailleurs réalisé un prototype reposant sur des composants peu onéreux (pointeur laser à 15 euros, WebCam et PC à 500 euros). Ils montrent que la technique laser demeure deux fois plus lente que celle de la souris $2D$ de bureau. Cependant, elle permet, d'une part, d'interagir à distance, et d'autre part, d'interagir à plusieurs sur un environnement graphique commun. Myers *et al.* [MBN+02] comparent la technique d'interaction par pointeur laser avec d'autres techniques : une souris classique, un assistant personnel tenu en main. Il en ressort que les techniques d'interaction classiques demeurent pour le moment plus efficaces, et les techniques laser, sous quelque forme qu'elles soient utilisées, sont assez imprécises et restent difficiles à maîtriser.

Peck *et al.* [Pec01] ont mené plusieurs expériences sur l'interaction à base de pointeur laser. Les données récoltées sont statistiquement analysées pour déterminer les meilleurs paramètres pour cibler un centre d'intérêt, diminuer le temps d'acquisition d'une cible, et enfin le temps pour déterminer une cible.

Les interfaces tangibles

Les interfaces tangibles ont pour but de réduire la séparation entre le monde virtuel et le monde réel [Bil01]. Pour cela, chaque objet virtuel est associé à un objet réel qui permet de le contrôler, et l'utilisateur interagit avec les objets virtuels en manipulant les objets réels (tangibles) correspondants. Par exemple, un classeur pourra être utilisé comme un menu dans lequel l'utilisateur range une série de feuilles physiquement vierges, mais associées virtuellement à un certain contenu.

Les interfaces tangibles, du latin *tangere* (toucher), cherchent donc à réaliser des interfaces intuitives dont la finalité est de coupler le réel et le numérique dans le but de simplifier l'interaction. Les applications des interfaces tangibles sont nombreuses, comme le système ActiveCube de Kitamura *et al.* [KIK01]. Garreau *et al.* [GC03] proposent d'utiliser les objets du monde réel comme outils d'interaction avec lesquels on pourrait manipuler des données numériques du monde virtuel associé. Certains auteurs ont utilisé des LEGO^{TM}que l'on emboîte

pour créer des formes particulières en réalité virtuelle. Le système ESKUA des auteurs est composé de 4 éléments : le cylindre, le parallélépipède, la vis, et le graft qui permet de multiplier les combinaisons possibles. ESKUA peut être par exemple utilisé pour la CAO (conception, alignement des pièces, ...).

Pangaro *et al.* [PMAI03] ont imaginé un périphérique magnétique à retour d'efforts un peu particulier. L'Actuated *Workbench* est un outil qui utilise la force magnétique pour bouger les objets sur une table, afin de fournir un retour à l'utilisateur. Parmi les utilisations possibles, on trouve les fonctions de recherche, de tri, annuler/refaire, mettre en valeur et guider les interventions de l'utilisateur. Sharlin *et al.* [SIW+02] utilisent des cubes dont les faces sont munies de capteurs et que l'on peut coller entre eux par magnétisme. Le principe est assez similaire à une construction de LEGO^{TM} : il suffit de les empiler les uns sur les autres, et la présence des aimants autorise de placer les cubes n'importe comment.

Les interfaces tangibles restent encore très expérimentales. Il arrive fréquemment que l'on ne comprenne pas très bien à quelle application elles sont destinées. Elles offrent néanmoins des perspectives et une alternative aux outils d'interaction traditionnels (souris $2D$ et $3D$, clavier, capteurs de position et d'orientation) intéressantes.

2.6.3 Commandes gestuelles et vocales

Les commandes gestuelles et vocales permettent une interaction naturelle et libre de l'utilisateur avec son environnement virtuel. Des mécanismes de reconnaissance sont nécessaires dans les deux cas, puisqu'il faut être en mesure d'associer une syntaxe et une sémantique aux différents gestes et paroles de l'utilisateur. La taille du dictionnaire et du vocabulaire disponibles dépend directement du nombre de commandes que l'utilisateur peut activer.

Nous allons successivement discuter du contrôle du système par commandes gestuelles, puis par commandes vocales. Dans les deux cas, nous commencerons par décrire le fonctionnement de ces techniques d'interaction, puis nous en donnerons les limitations, avant de conclure sur certaines applications de réalité virtuelle qui tirent parti des possibilités qu'elles offrent pour le contrôle du système.

Commandes gestuelles

Les gestes constituent l'une des premières techniques à avoir être employée pour le contrôle du système en environnement $3D$. Les chercheurs souhaitant rapprocher l'homme et la machine, il semblait évident que les mains puissent jouer un rôle essentiel dans cette tâche, d'autant que l'absence de périphérique est un argument supplémentaire. Les commandes gestuelles se partagent en deux groupes distincts : d'une part les postures, et d'autres part les gestes.

Une posture est typiquement une configuration statique de la main ; même si l'utilisateur choisit de bouger l'ensemble de sa main dans l'espace, l'essentiel est que les phalanges ne bougent pas entre elles. Par exemple, lorsque l'on garde le

poing fermé pour attraper quelque chose, il s'agit d'une posture. À l'inverse, un geste est un mouvement dynamique de la main, c'est-à-dire que la configuration des doigts et du poignet changent. Par exemple, on pourra utiliser son index pour écrire un mot dans l'espace.

Bien entendu, les limites du contrôle du système à l'aide des postures et des gestes sont à chercher dans le nombre et la complexité des commandes gestuelles qui peuvent être reconnus. De surcroît, plus il y a de postures et gestes disponibles et plus la charge cognitive d'apprentissage et d'utilisation sera élevée, et demandera du temps.

Une des applications les plus intéressantes à avoir tiré parti de l'utilisation des gestes est Polyshop de Mapes et Moshell [MM95]. Ces dernières imaginent une interaction complète basée sur l'utilisation des mains et des gants de données, depuis la navigation jusqu'aux menus. Par exemple, lorsque l'utilisateur souhaite se déplacer, il lui suffit de faire un mouvement de pincement avec ses doigts, puis de tirer ou pousser selon qu'il souhaite se rapprocher ou s'éloigner, en « s'accrochant » littéralement à l'espace de travail virtuel. Pour dessiner, il n'est pas nécessaire de préciser que l'on change de mode, puisque celui-ci est implicite en changeant de geste : il suffit de mettre la main à plat puis de la bouger dans l'espace pour dessiner des courbes. D'autres applications à interaction gestuelle ont vu le jour, comme par exemple SKETCH de Zeleznik *et al.* [ZHH96] et Teddy d'Igarashi *et al.* [IMT99] qui permettent de dessiner des formes libres dans un espace $3D$.

Commandes vocales

La reconnaissance vocale peut-être utilisée pour deux types d'interaction. D'une part passer des commandes au système, et d'autre part entrer du texte. Nous développons ici le premier point, le second faisant l'objet d'une discussion en section 2.7 sur l'entrée de symboles. Passer une commande à l'aide de la voix représente, au même titre qu'une commande gestuelle, une manière très intéressante, très naturelle et intuitive d'interagir. La voix, en tant que support de communication réduit le nombre de périphériques intermédiaires entre l'homme et la machine, et l'accès à cette manière de faire est immédiat pour l'utilisateur.

Le moteur de reconnaissance vocale est l'élément le plus critique dans un système à commandes vocales, car un large éventail de facteurs influence le taux de réussite et la vitesse d'acquisition des commandes. En effet, plusieurs utilisateurs différents pourront se servir de ce paradigme, alors qu'ils ont des voix complètement différentes et que l'environnement sonore peut lui aussi varier fortement, notamment lorsqu'il y a du bruit. Lorsque le moteur de reconnaissance est dépendant de chaque utilisateur, il faut préalablement une phase d'entraînement et concevoir un profil unique pour chaque participant. À l'inverse, un système indépendant des utilisateurs ne requiert aucune préparation spécifique. Précisons que les capacités d'un système de reconnaissance dépend fortement de la taille de son dictionnaire et du vocabulaire qu'il contient. Plusieurs produits de reconnaissance existent, comme ViaVoice d'IBM ou l'Unisys Natural Language Speech Assistant.

La particularité de ce paradigme de contrôle est d'être totalement transparent. Il est donc important que les fonctionnalités de l'interface vocale soient connues de l'utilisateur pour être utilisées. De plus, pour renseigner l'utilisateur sur le résultat de la commande, il faut y adjoindre un retour d'informations pour signifier qu'elle a échoué ou au contraire réussi. À l'inverse d'une métaphore de contrôle comme le menu, la reconnaissance vocale est active en permanence, sauf lorsqu'il y a un changement de mode explicite. Précisons également que lorsque plusieurs personnes travaillent dans le même environnement, il faut que le système soit capable de faire la distinction entre le flux de paroles entre utilisateurs et les commandes vocales qui lui sont adressées.

2.7 L'entrée de symboles

Alors que nous sommes habitués au clavier et à la souris, il est étrange de constater que, dans la majorité des situations, les environnements de réalité virtuelle ne proposent que peu de solutions pour entrer des caractères, des chiffres, et plus globalement des symboles dans la machine. Pourtant, il s'agit là d'une tâche fondamentale d'interaction. À l'inverse, le monde du *personnal computer* a commencé par fournir les outils propres aux traitements de textes. Puis, beaucoup plus récemment des paradigmes d'interaction pour manipuler des objets graphiques en $3D$.

En fait, on pourrait presque parler d'accessibilité, puisque c'est suivant le matériel disponible et les conditions de travail que l'utilisateur sera en mesure de réaliser telle ou telle tâche. Par exemple, dans un CAVE, l'utilisateur est debout, n'a aucun support pour poser ni clavier, ni souris, ni même ses bras. Actuellement, il dispose d'une large gamme d'outils pour créer une sculpture de toute pièce mais rien pour enregistrer sa création dans un fichier informatique.

Dans cette partie, nous allons voir qu'il existe 3 classes de techniques d'interaction pour entrer des symboles : les claviers, qui s'inscrivent dans un continuum avec les environnements de bureau, la reconnaissance de gestes et la reconnaissance vocale.

2.7.1 Les claviers

Le clavier d'environnement de bureau

Les premiers claviers ont été inventés pendant le 18^{e} siècle et le mode *qwerty* a été introduit dès 1873 par Christopher Sholes, en se basant sur la fréquence des lettres dans un texte rédigé en alphabet latin. Très rapidement, le clavier *qwerty* a été adapté au Français, avec ses lettres particulières (accents, cédilles, ...) en configuration *azerty*. Bien que d'autres méthodes aient été testées avec plus ou moins de succès, le clavier demeure le moyen d'entrer des symboles le plus courant en informatique. Lorsque l'utilisateur est assis devant son écran, et qu'il dispose d'un support pour poser son clavier et ses avant-bras, il est aisé de taper de longs textes ou de combiner plusieurs touches entre elles. En revanche, dans un environnement immersif, où il est debout, le clavier traditionnel n'est plus adapté. Quelques tentatives visent à améliorer l'ergonomie des claviers pour les rendre portatifs.

Le clavier miniature

Disposer d'un clavier miniature permet de pouvoir le fixer directement sur l'utilisateur, par exemple sur l'avant bras. Cependant, la largeur des touches ne doit pas descendre en-deçà d'une certaine taille, relative à la grosseur des doigts. C'est notamment la principale critique émise à l'encontre des téléphones portables aujourd'hui. Hormis la taille des touches, leur nombre joue également un rôle déterminant. On trouve désormais des claviers à 120 touches, trop pour une utilisation même miniaturisée. Dans ce cas, pourquoi ne pas combiner les touches entre elles pour produire toute sorte de symboles ?

Modifier le matériel est une chose, assister l'utilisateur lors de sa frappe au clavier en est une autre. Sur les téléphones portables, les constructeurs ont mis en place le langage *T9*, véritable aide à l'écriture. Cette technique s'avère rapide lorsque le mot est connu, mais il faut la désactiver lorsqu'on souhaite taper une suite de symboles qui n'ont *a priori* aucun sens, comme une abréviation.

Le clavier virtuel

En lieu et place d'un clavier physique, une autre idée consiste à projeter une image de clavier sur une surface, puis à reconnaître les mouvements des doigts pour savoir ceux qui sont entrés en contact avec les touches affichées. La société Canesta a proposé un clavier virtuel un peu particulier. En effet, cette start-up a développé un système qui utilise un mini-laser de la taille d'un grain de café pour projeter l'image d'un clavier à une distance de 50cm, avec un angle de 50 degrés, le tout sur une surface plane. L'image du clavier mesure 28cm de longueur pour environ 10cm de largeur, soit l'équivalent des claviers détachables commercialisés aujourd'hui pour les assistants personnels. Une micro-caméra intégrée détermine ensuite les touches frappées par l'utilisateur.

Le Senseboard Virtual Keyboard de Senseboard est un autre clavier virtuel qui repose sur le même concept. *A priori* n'importe quelle surface plane peut être utilisée, cependant plutôt que de suivre les mouvements des doigts *via* une caméra, ce périphérique enregistre les variations musculaires des mains à l'aide de capteurs de pression disposés sur le dessus et la paume des mains. L'inconvénient des claviers virtuels est leur manque de retour d'informations, puisque l'utilisateur n'a pas la sensation d'appuyer sur une touche. De plus, s'ils peuvent convenir dans un environnement de type *Workbench* où l'on trouve une surface plane, dans un CAVE, par exemple, il est difficilement imaginable d'utiliser de tels périphériques.

2.7.2 La reconnaissance de gestes

La reconnaissance des gestes est une forme d'interaction assez intéressante en réalité virtuelle, puisqu'elle se base sur l'utilisation des mains, forme d'interaction naturelle par excellence. La plupart du temps on utilise comme périphériques les gants de données combinés au suivi des mouvements des mains et des phalanges. Une phase de calibrage est souvent requise, selon la morphologie de chacun. Dans le cas de l'utilisation des gestes pour entrer des symboles, on peut envisager deux solutions. D'une part reconnaître un langage de signes, et d'autre

part reconnaître l'écriture produite par les doigts dans l'air ou au contact d'une surface, c'est-à-dire le dessin de courbes.

Les langages de signes

Il existe dans le monde une quantité conséquente de langage des signes, différents selon la langue et le domaine d'application. Basés à la fois sur la reconnaissance des formes de la main, des doigts et des mouvements, les langages basés signes permettent de se parler très rapidement. Une tentative de [FH98], à l'aide du système GloveTalk, a permis de traduire des signes en sons. Les signes sont alors reconnus par un réseau de neurones que l'on a préalablement entraîné.

Parmi les langages de signes, certains appartiennent à certaines branches professionnelles (alphabet sémaphore des marins). Cependant, cette façon de procéder limite le nombre d'actions possibles à la liste de gestes du domaine. Il est peu probable que l'alphabet des plongeurs soit adapté à la modélisation géométrique, par exemple on peut baser le langage sur des signes couramment utilisés par tout le monde, pour limiter le temps d'apprentissage. En effet, plus le nombre d'actions disponibles – et donc de gestes associés – augmente, et plus le temps d'apprentissage est long. Qui plus est, lorsque les gestes sont trop proches ils peuvent dans une certaine mesure se parasiter les uns les autres, car le système les confondra.

Le plus souvent, la reconnaissance de gestes utilise des gants de données munis d'un capteur de position et d'orientation, ainsi que des capteurs précisant les angles des phalanges et du reste des doigts. On peut donc déterminer plus ou moins finement quel signe a été fait par l'utilisateur. Certains gants disposent de capacités supplémentaires, comme par exemple les Pinch Gloves de la société Fakespace. Il s'agit avant tout de gants de commande sur lesquels sont détectés les contacts entre les doigts. On peut ainsi se servir de chaque gant comme d'une liste de commandes possibles. [BLC01] utilise ces gants pour simuler un clavier *qwerty*. Le clavier virtuel est affiché à l'écran sous forme d'un alphabet rangé dans un tableau. Chaque doigt permet de changer de colonne, et le rapprochement des mains près du corps permet de changer de ligne. Enfin chaque rotation du poignet valide la lettre sélectionnée. Ce paradigme d'interaction est à la frontière de la reconnaissance de gestes et des claviers.

L'écriture et le dessin

L'écriture manuscrite, basée sur l'utilisation d'un crayon, d'un stylo ou encore d'un pinceau (calligraphie chinoise par exemple) semble être sur le déclin, et en passe d'être remplacée par le "tout clavier". Cependant, l'emploi du manuscrit peut également servir à entrer des symboles sur l'outil informatique. Une des premières tentatives a été réalisée chez Palm ([Com95]). En effet, sur les assistants personnels, l'idée est de fournir un ordinateur de poche sans clavier, tout en autorisant les entrée de textes et dessins. Il suffit d'utiliser un stylet sur un écran tactile en traçant des lettres d'un alphabet simplifié (l'alphabet Graffiti 2). Bien que n'étant pas de la reconnaissance directe de gestes, nous considérons cependant que le lien qui existe entre les langages de signes et l'écriture et le dessin est très fort.

L'écriture manuscrite est basée sur un ensemble de lignes courbes, tracées sur une surface d'ordinaire plane. Lorsqu'elles sont affichées, elles aident l'utilisateur à savoir ce qu'il est en train d'écrire et ce qu'il a déjà écrit, s'il fait des fautes, si l'écriture est droite, etc. Il faut donc que le système reconnaisse toutes ces lignes pour déterminer quel symbole doit y être associé, en fonction du dictionnaire enregistré dans la mémoire de la machine.

Les systèmes Graffiti et Graffiti 2, mis au point par la société 3Com et utilisés sur Palm sont basés sur des alphabets dont la forme des symboles est simplifiée par rapport à l'alphabet latin. Toutefois, la manière de les écrire s'en rapproche suffisamment pour que l'utilisateur ne soit pas déboussolé. D'autres techniques similaires ont été imaginées, comme le Quickwriting de Perlin [Per98]. Comme pour Graffiti, l'entrée de symboles s'effectue à l'aide d'un stylet. Cependant, au lieu d'imiter la forme d'une lettre, l'alphabet est affiché concentriquement à la manière d'un *Marking Menu*. Pour sélectionner un caractère il suffit d'en donner la direction.

[MA98] présentent le système Cirrin, qui s'inspire fortement du Quickwriting de Perlin, à la différence qu'ici les lettres sont placées autour d'un cercle, que l'utilisateur parcourt jusqu'à ce qu'il estime avoir terminé. Il revient alors vers le centre du cercle en position neutre et lève le stylet pour terminer un mot. À la place d'un assistant personnel, [PTW98] ont proposé le Virtual Notepad qui se compose d'un stylet et d'une tablette tactile associée à un logiciel de reconnaissance d'écriture. Ici, l'utilisateur écrit comme il le fait tous les jours, sans symboles particuliers.

Outre la reconnaissance des symboles connus, il peut être intéressant de dessiner des formes que l'ordinateur n'a pas besoin de connaître (ex : annotations, signatures, ...). Dès lors, la machine ne pourra pas déterminer la sémantique du dessin, et il sera donc enregistré comme simple fichier graphique dans la mémoire de la machine.

2.7.3 La reconnaissance vocale

Un ordinateur est en mesure de décoder notre locution pour effectuer certaines tâches. Depuis 1945, de nombreux chercheurs se sont intéressés à ce mode de communication entre l'homme et la machine. En pratique, lorsque le son a été émis par le locuteur, il est capté par un microphone, puis numérisé à l'aide d'un convertisseur analogique-numérique. Le signal peut éventuellement être compressé avant que son analyse ne commence (transformée rapide de Fourrier, méthode d'identification, analyse fractale, ...).

Théoriquement, la reconnaissance vocale présente un certain nombre d'avantages. Elle rend les membres du corps totalement disponibles pour d'autres tâches, et permet d'entrer de grandes quantités de texte, plus rapidement qu'avec un clavier : on parle plus vite qu'on ne tape. C'est une technique plus confortable pour l'utilisateur car il l'utilise tous les jours de façon naturelle. Il ne faut pas oublier qu'elle est souvent présentée dans les médias comme la solution d'avenir pour interagir avec les outils informatiques. Pour le moment, dans les faits,

l'efficacité n'atteint pas encore celle de certains films de science-fiction, même si des progrès considérables ont été réalisés dans ce domaine.

Les problèmes sont avant tout liés à la technologie employée pour la reconnaissance vocale. Elle s'avère à l'heure actuelle peu précise, lente et nécessite de l'entraînement. De plus, des sources sonores externes peuvent venir parasiter le son de la voix. Enfin, il convient de prendre en compte le but visé par la reconnaissance de la parole. On l'utilise soit dans le but de passer des commandes à l'application – ou de manière plus générale au système –, soit pour éditer du texte et d'autres symboles comme on pourrait le faire à l'aide d'un traitement de texte et d'un clavier.

Reconnaissance d'un caractère

Reconnaître un caractère est la chose la plus simple en reconnaissance vocale. En effet, il s'agit de la brique élémentaire à partir de laquelle on construit les mots, eux-mêmes étant assemblés pour former des phrases. Et la difficulté réside avant tout dans la séparation des différents éléments sonores. Les caractères permettent de rentrer des données lettre par lettre, sans que le système n'ait à vérifier la cohérence des données produites. C'est en effet le rôle de l'utilisateur.

Deux systèmes existent pour l'instant. Le *monolocuteur* repose sur le principe qu'un utilisateur unique se sert du système pour dicter un ensemble de caractères. Le taux de reconnaissance est donc assez important et il est tout à fait possible d'étendre le vocabulaire utilisable. Lorsque plusieurs personnes doivent utiliser le même système, on le dit *multilocuteur*. Il contient une base de données comprenant des empreintes moyennes, les taux de reconnaissance sont moins bons qu'en mode monolocuteur et le nombre de caractères plus limité.

La reconnaissance caractère par caractère reste de toute façon trop longue lorsqu'il s'agit d'entrer de nombreuses lettres, malgré un taux de reconnaissance plus important que pour les méthodes que nous allons voir à présent.

Reconnaissance d'une chaîne de caractères par dictionnaire

La reconnaissance caractère par caractère est efficace, mais très lente. En utilisant comme brique de base des mots complets, il est possible d'accélérer le processus. Par contre, le nombre d'erreurs risque également d'augmenter : d'une part les empreintes sonores deviennent plus complexes à analyser et, d'autre part, lorsque l'utilisateur conjugue un verbe par exemple, le système de reconnaissance peut se tromper de par la proximité sonore de deux mots.

Le nombre de mots que peut analyser le système dépend de la taille du dictionnaire, c'est-à-dire de la quantité de vocabulaire dont il dispose. Lorsque celui-ci est trop grand, on recourt à la méthode analytique : les mots ne sont plus mémorisés dans leur intégralité, mais traités en tant que suite de phonèmes et de syllabes. Le dictionnaire est alors composé non plus de mots, mais de syllabes et de phonèmes.

Ainsi lorsque la taille du vocabulaire est peu importante, on choisira une approche globale (basée sur un dictionnaire de mots). À l'inverse, pour un grand dictionnaire, on choisira la méthode analytique. Ces deux techniques ne présentent pas non plus le même coût : l'analytique est plus onéreuse en terme de calculs, car elle repose sur l'utilisation de gros réseaux de neurones ou de chaînes de Markov cachées. Enfin, précisons que cette dernière est bien plus efficace lorsque les mots s'enchaînent rapidement (c'est-à-dire lorsque le blanc sonore de séparation entre les mots est court).

Reconnaissance d'une chaîne de caractères quelconque

Lorsque l'on souhaite entrer une chaîne de caractères dont la signification n'est pas particulièrement importante, on considère alors qu'il s'agit d'un mode de prises de notes vocales. Le système de [HPRB96] propose d'attacher des notes sonores aux objets virtuels de la scène. Il est possible d'enregistrer et d'écouter des annotations sonores, ainsi la vue de l'utilisateur n'est pas encombrée par une note textuelle.

2.8 Conclusion

Nous venons de voir les principaux paradigmes d'interaction que l'on rencontre en réalité virtuelle. Cet univers est loin d'être figé, chaque jour de nouvelles techniques voient le jour ; d'une part parce que le matériel évolue sans cesse ; d'autre part parce qu'il y a des besoins applicatifs spécifiques. L'interaction exploite principalement le sens visuel mais d'autres modalités sensorielles sont maintenant activement étudiées ([TBBB02]), comme par exemple le toucher avec le retour d'effort.

Les progrès sont nombreux, mais malgré tout, de nombreux problèmes demeurent. La faute aux limites du matériel actuel ? Pas seulement. Les limitations proviennent également de l'usage que nous faisons de nos environnements virtuels. En terme d'outils de développement, ou d'aide au développement, il n'existe pas grand chose. Chaque programmeur doit souvent repartir d'une feuille blanche ou presque, braver les embûches logicielles posées ça et là par une kyrielle de librairies qui n'ont pas été pensées pour fonctionner de concert. La solution unique et standardisée, comme le sont la souris et le clavier sur nos environnements de bureau, n'existe pas encore, et n'existera vraisemblablement jamais.

L'élaboration d'un outil d'interaction complet doit se voir dans sa globalité ; dès lors, l'utilisateur devrait avoir accès à des paradigmes certes plus complets, mais pas plus complexes. Intuitifs, efficaces, adaptés à la tâche, ergonomiques sont des des exigences souvent exprimées par les utilisateurs. Aujourd'hui, des outils d'interaction d'un nouveau genre commencent à montrer le bout du nez sur nos environnements de bureau ; ils s'adaptent automatiquement à chaque utilisateur, en fonction de ses comportements. Pourquoi ne pas imaginer un parallèle sur les environnements de réalité virtuelle ?

La réalité virtuelle de demain devrait prendre de plus en plus d'importance et

intéresser un public toujours plus large. Certes encore très onéreux, les environnements immersifs sont d'ores et déjà très utilisés dans des domaines très divers. Ils ont fait la preuve de leur efficacité, par exemple dans l'industrie automobile sur le processus de prototypage d'une voiture. Ou encore en psychiatrie, où les résultats de traitement de certaines pathologies donnent à réfléchir, comme le traitement de l'agoraphobie (peur de la foule) en plaçant le patient dans un environnement virtuel urbain parfaitement contrôlé. L'interaction n'en est encore qu'à ses débuts, mais déjà les résultats sont très encourageants.

Chapitre 3

Les interfaces matérielles

3.1 Introduction

L'être humain a créé la machine. Pourtant, dès le début, il a éprouvé beaucoup de difficultés à dialoguer avec sa création. Aujourd'hui de nombreuses interfaces permettent à l'homme d'interagir sur la machine et vice-versa. Nous allons voir quel matériel est utilisé dans les laboratoires ou les entreprises pour plonger l'utilisateur dans un monde virtuel. L'intention première d'un tel environnement étant de pouvoir interagir d'une manière naturelle et multimodale. Beaucoup d'interfaces sont encore expérimentales, voire artisanales, et il faudra du temps pour qu'elles entrent chez le particulier. Pour le moment leur coût reste trop élevé, et la technologie trop immature. Le manque d'uniformisation et de standards au niveau du matériel freine le développement de la surcouche logicielle, préalable – elle-même à standardiser – nécessaire à la démocratisation de la réalité virtuelle.

Nous nous intéresserons premièrement aux interfaces visuelles (section 3.2), qui permettent de stimuler la vision. Comme nous l'avons vu dans le chapitre 1, ce sens monopolise près de 70% de l'attention de l'utilisateur. La recherche sur le matériel dédié à cette sensibilité sensorielle est importante, d'autant qu'avec le cinéma, la télévision et aujourd'hui les nouveaux périphériques portatifs tel le téléphone portable et les assistants personnels, on dispose de nombreux équipement différents, répandus, abordables, efficaces et bien connus. Dans ce domaine la recherche actuelle porte essentiellement sur la stéréovision, l'holographie, le rendement et la miniaturisation du matériel.

Dans un deuxième temps (section 3.3), nous survolerons de manière non exhaustive les dispositifs d'interaction permettant de suivre les mouvements du corps de l'utilisateur dans l'espace. Les capteurs de position et d'orientation, aussi nommés traqueurs, seront abordés et expliqués. Puis, dans la section 3.4 nous verrons les périphériques de pointage $3D$ et les gants de données, regroupés sous la notion d'interfaces d'entrée, qui permettent de manipuler le monde virtuel et les objets qui s'y trouvent. Les interfaces haptiques, que l'on présentera en section 3.5, stimulent le sens du toucher et fournissent un retour tactile et kinesthésique. Elles servent notamment à contraindre les mouvements de l'utilisateur dans l'espace.

Enfin, la dernière section de ce chapitre traitera des interfaces de stimulation de l'ouïe, de l'odorat et du goût. Nous notons que l'ouïe, bien que comptant pour 20% de l'information sensorielle chez l'homme, est assez peu employée en réalité virtuelle, notamment en comparaison des interfaces haptiques. Ce qui est d'autant plus étrange, étant donné que le matériel dédié à la spatialisation du son existe depuis longtemps et fonctionne assez bien. Concernant l'odorat et le goût, les recherches n'ont réellement débuté qu'il y a 10 ans environ. Peu de matériel existe aujourd'hui pour ces deux derniers sens, et leur utilité en réalité virtuelle n'est pas encore bien connue.

3.2 Interfaces visuelles

Les interfaces visuelles constituent le composant principal de l'interaction entre l'homme et la machine en réalité virtuelle. Elles sont là pour stimuler le sens de la vue, car elles permettent d'afficher une représentation visuelle du monde virtuel. Il existe une grande variété de dispositifs d'affichage, que nous nommons interfaces visuelles. Dans les paragraphes qui suivent, nous allons décrire plusieurs de ces périphériques, en indiquant les principaux avantages et inconvénients de chacun. Nous voyons d'abord les interfaces fixes, dont le système d'affichage des images ne peut être déplacé, puis les dispositifs portables, dont le système d'affichage est contenu dans des lunettes spéciales.

3.2.1 Interfaces fixes

L'interface la plus accessible reste l'**écran** à tube cathodique ou à cristaux liquides. Puisque chaque ordinateur dispose généralement d'un moniteur, il n'y a pas besoin d'acheter et de configurer un nouvel équipement pour afficher un environnement virtuel. Ces interfaces disposent d'une bonne résolution et d'un large éventail de couleurs. Pour activer la vision stéréoscopique, il faut que l'utilisateur porte des lunettes[1] qui pourront obturer successivement la vision de chaque œil. On appelle généralement ce type d'interface un *fish tank* [ABW93], car on a l'impression de regarder à l'intérieur d'un aquarium du fait de la taille du moniteur. Le problème principal de cette interface est sa taille relativement petite. Les utilisateurs n'ont pas la sensation de faire partie du monde virtuel ; ils ont simplement l'impression de regarder par une fenêtre dans le monde virtuel. Un autre problème, lui aussi relatif à la taille du moniteur, réside dans le fait qu'il est la plupart du temps impossible d'explorer des objets virtuels affichés à l'échelle 1 : 1. Enfin, des problèmes de perception de profondeur apparaissent sur les objets qui sont affichés au bord de l'écran : l'utilisateur a la sensation qu'ils se trouvent à la même profondeur que l'écran, et non pas devant ou loin derrière.

Le **plan de travail virtuel**[2] offre une alternative intéressante aux moniteurs. Les dispositifs comme le BARON^{TM}(figure 3.1), le VersaBench^{TM}ou l'Immersadesk^{TM}sont des écrans dont la taille est quasiment celle d'un bureau (1m20 par 1m50 et inclinés à 45 ° dans le cas de l'Immersadesk^{TM}). Ces plans de travail offrent un environnement de type semi-immersif. C'est-à-dire que les objets n'entourent pas l'utilisateur comme dans un CAVE par exemple. Par

[1] éventuellement associées à un capteur de position et d'orientation.
[2] a. *desk*

rapport à un moniteur, un plan de travail autorise toutefois une meilleure sensation d'immersion, sans bloquer complètement la vue du monde physique. Ce type d'affichage s'avère tout à fait indiqué pour les applications scientifiques ou les simulations d'opérations médicales. Le chirurgien pourra par exemple s'entraîner sur des patients virtuels avant une opération réelle. De plus, il est possible d'incliner le plan, ce qui offre un effet de profondeur particulier. Enfin, ce dispositif présente l'avantage de prendre assez peu de place et d'être relativement transportable.

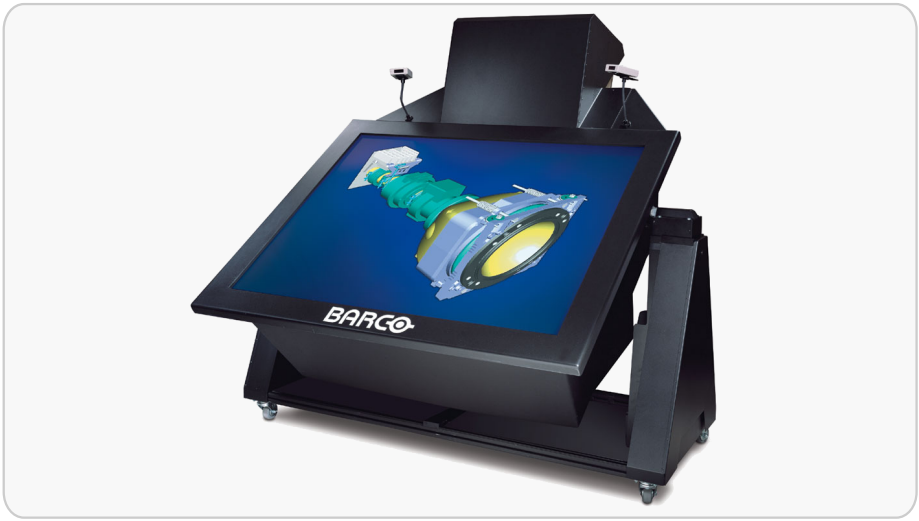


Fig. 3.1: Plan de travail Baron^{TM}de Barco.

À mi-chemin entre les plans de travail et les environnements à base de projection que nous allons voir dans le paragraphe suivant, on trouve le VisionStation d'Elumens (figure 3.2) qui est un système d'exposition hémisphérique semi-immersif, dont le champ de vision est de 180 °. L'utilisateur se positionne à la frontière entre le monde physique et le monde virtuel, et dispose d'un clavier et d'une souris.

Une autre technique d'affichage repose sur une technologie qui existe depuis un certain nombre d'années : la **projection**. D'ordinaire, les systèmes projectifs sont employés pour afficher des informations à un groupe d'individu (par exemple au cinéma). Ils possèdent certains avantages par rapport aux moniteurs ou aux plans de travail immersifs. Leur taille est bien plus importante, offrant potentiellement une meilleure immersion. Il est alors possible d'afficher et de naviguer dans un monde virtuel à l'échelle de l'homme. Ces dispositifs d'affichage existent sous différents formats.

La configuration la plus simple est un écran unique, qui élargit le champ de vision de l'utilisateur. Cependant, ce dernier peut encore avoir la sensation d'être en dehors du monde virtuel. Par exemple, lorsqu'il tourne sur lui-même, il ne voit plus rien de la scène virtuelle, puisqu'il n'y a aucun écran derrière lui.

Fig. 3.2: Système hémisphérique VisionStation^{TM}d'Elumens.

Le système CAVE^{TM}[CNSD93] tend à résoudre le problème du simple mur immersif. Pour cela, tous les murs d'une pièce, ainsi que le sol et le plafond servent d'écran. Les utilisateurs entrent dans ce cube par une petite porte, et peuvent tenir en général à 8 ou 10 dans une même pièce. Les CAVE ne disposent pas tous de 6 murs (de 2 à 5 mètres de côté) sur lesquels est projeté l'environnement virtuel. Des variantes à 2, 3, 4 ou 5 murs existent, par exemple le système MoVE^{TM}à 3 murs de Barco (figure 3.3). Dans ce type d'interfaces visuelles, la sensation d'immersion est bien meilleure par rapport aux autres systèmes fixes. Ils sont particulièrement bien adaptés pour des tâches de navigation. Par contre, ils sont très onéreux et complexes à entretenir, et nécessitent un espace très volumineux (au moins 6 mètres en hauteur dans le cas d'un système à 6 écrans).

Certains chercheurs travaillent actuellement à la mise au point d'écrans de nouvelle génération. Une équipe de la société Sharp a créé un écran permettant d'afficher des images en $3D$, tout en s'affranchissant des lunettes obturatrices. L'affichage des images se fait à travers une grille de lentilles orientées vers les deux yeux de l'utilisateur : chaque œil perçoit donc une image différente. Le principal obstacle rencontré sur cette technologie est que l'utilisateur doit rester dans un certain périmètre devant son écran, afin que la convergence des images sur ses yeux reste correcte. Pour le moment, ces écrans sont encore très fatigants pour les yeux et donnent mal à la tête.

L'écran holographique constitue une autre approche de l'affichage d'objets $3D$. Dans le cas d'une dalle holographique, un écran souple et quasi-transparent est spécialement étudié pour capturer la luminosité d'une projection arrière. Le même procédé est utilisé sur des écrans à fumée : de fines particules en suspension capturent la lumière de deux faisceaux laser qui se croisent en un point. Très expérimentaux, ces dispositifs ne sont pour l'instant pas suffisamment lu-

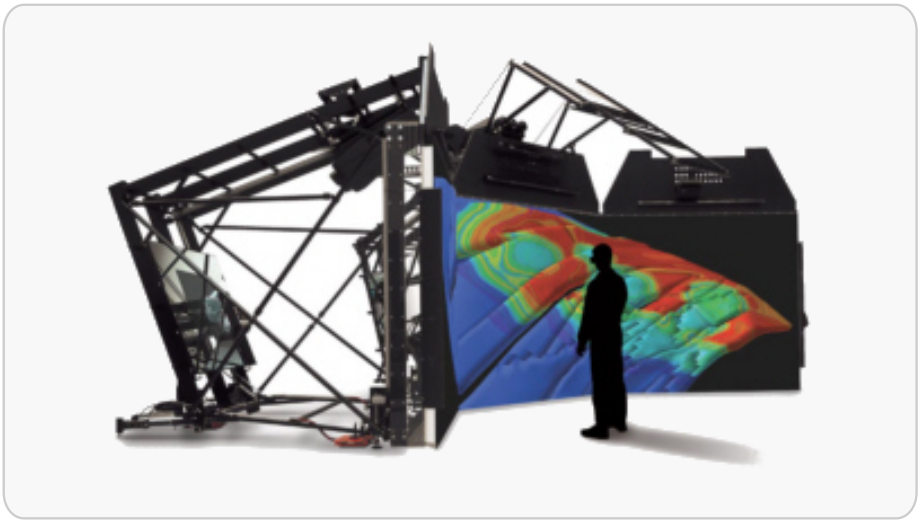


Fig. 3.3: Système MoVE^{TM}à 3 murs de Barco.

mineux, et nécessitent la combinaison de plusieurs lasers de couleur différente pour afficher une image volumique polychromatique.

3.2.2 Interfaces portables

Les interfaces portables ont été les premiers dispositifs à avoir été utilisés en réalité virtuelle. En 1962, Sutherland invente le premier casque monoscopique et il faut attendre le début des années 70 pour qu'apparaissent les premiers systèmes portables stéréoscopiques. Les systèmes portatifs de cette période initiale ont abouti aujourd'hui à plusieurs variantes que nous allons survoler.

Les premiers systèmes utilisés étaient les visio-casques ou *head mounted display* (HMD) en anglais. Il s'agit de lunettes dont les verres ont été remplacés par deux écrans (un par œil). Le mouvement de la tête de l'utilisateur est enregistré en permanence et permet de calculer une image dépendant de la position de son point de vue. Le système Glasstron^{TM}de Sony, figure 3.4, est un des premiers casques portatifs à faible coût qui a été disponible sur le marché. Ces dispositifs souffrent d'une résolution assez faible, et d'un champ de vision assez restreint. Lorsque l'utilisateur ne voit que le monde virtuel, il ne peut distinguer son propre corps physique, ce qui nuit à la sensation d'immersion. Pour atténuer ce problème, on introduit la notion d'avatar ou corps virtuel, qui est une représentation virtuelle de tout ou d'une partie du corps physique. En outre, l'utilisateur doit être surveillé en permanence : il ne voit pas où il marche en réalité et pourrait se blesser. D'autres problèmes peuvent survenir, comme par exemple le poids du système qui limite le temps d'utilisation et engendre parfois une fatigue au niveau du cou.

Il existe de nouvelles techniques d'affichage des images qui remplacent les

Fig. 3.4: Système HMD Glasstron^{TM}de Sony.

écrans et les lunettes. On peut par exemple citer le système d'affichage rétinien[3] qui se base sur un rayon laser qui balaye le fond de la rétine pour y afficher une image. Les avantages sont nombreux, comme par exemple la taille, le poids, la largeur du champ de vision supérieur à 120° et la résolution très élevée proche de la vision humaine. Ce type de dispositif permet en outre d'obtenir un véritable affichage stéréoscopique avec modulation de la profondeur. Certains systèmes comme le Nomad^{TM}de Microvision (figure 3.5) n'emploient qu'un seul faisceau laser et sont donc monochromatiques. Mais des versions à trois couleurs mélangées sont à l'essai. Ces interfaces sont encore expérimentales et pour le moment, le nombre de couleur est assez limité, puisqu'il dépend du type de laser qui frappe le fond de la rétine.

Fig. 3.5: Système d'affichage laser rétinien Nomad^{TM}de Microvision.

3.3 Interfaces de suivi du mouvement

L'interaction avec un environnement virtuel requiert au minimum deux informations : la position et l'orientation de la tête de l'utilisateur, afin que le point

[3]a. *Virtual Retinal Diplay* ou *VRD*

de vue puisse être convenablement calculé. D'autres zones du corps peuvent être suivies dans l'espace, comme par exemple les mains ou le torse. Les objets tridimensionnels disposent de 6 degrés de liberté[4] : leur position dans l'espace (x, y et z) ainsi que leur orientation (angles de tangage, roulis et lacet). Pour manipuler ces objets, on peut employer des traqueurs[5] à 6 degrés de liberté. Il est également possible de restreindre le nombre de degrés de liberté pour contraindre un déplacement sur un axe par exemple. Il existe trois types de traqueurs :

- ceux qui fonctionnent en **mode absolu** : ils fournissent des données de position et d'orientation par rapport au repère global de la scène ;
- ceux qui fonctionnent en **mode relatif** : ils fournissent des données de position et d'orientation par rapport au repère local à un autre objet ;
- ceux qui fonctionnent **sans référence** : ils fournissent des données d'orientation (assiette, tangage et roulis), ainsi que l'accélération angulaire, la dérive et la vitesse.

De façon générale, le suivi du mouvement[6], également nommé suivi de la position et de l'orientation, est employé en environnement virtuel pour suivre dans l'espace la position et l'orientation d'objets physiques. En fonction de chaque application, 5 critères permettent de choisir le traqueur le plus adapté [HL93] :

- **taux de rafraîchissement** : définit combien de mesures par seconde sont réalisées (mesuré en Hz). Plus ce taux est important et plus les mouvements liés au traqueur seront fluides, ce qui implique aussi plus de traitement et de temps de calcul.
- **latence** : correspond au temps (mesuré en ms) entre l'action physique de l'utilisateur et le déplacement effectif d'un objet virtuel par exemple. Les temps les plus faibles contribuent aux meilleurs performances.
- **précision** : les mesures de position et d'orientation sont toujours erronées, puisque le traqueur fournit des valeurs qui sont des approximations de la position et de l'orientation correctes du traqueur dans l'espace. Les valeurs d'erreur les plus faibles correspondent aux traqueurs les plus précis.
- **résolution** : il s'agit des plus petits changements de valeurs de position et d'orientation que le traqueur est capable d'enregistrer. Plus ces valeurs sont faibles, plus la précision augmente.
- **espace de travail** : correspond au volume à l'intérieur duquel le traqueur est utilisable, avec des critères de résolution, d'orientation, de précision, de latence et de taux de rafraîchissement convenables.

Différents types de traqueurs existent et se basent sur des technologies différentes, chacune ayant ses forces et ses faiblesses. Nous ne parlerons pas du suivi des yeux, qui fonctionne différemment : on mesure la direction vers laquelle pointent les yeux de l'utilisateur. Les cinq sections qui suivent détaillent le fonctionnement des traqueurs magnétiques, acoustiques, optiques, mécaniques et inertiels ; les trois premiers sont les plus employés en environnement virtuel.

[4] a. *degrees of freedom* ou *DOF*
[5] Un traqueur est capteur de position et d'orientation (a. *tracker*).
[6] a. *Tracking* ou encore *Position and Orientation Tracking*

3.3.1 Traqueurs magnétiques

Les traqueurs magnétiques sont certainement les plus utilisés en réalité virtuelle. Ils fonctionnent sur un principe assez simple. Plusieurs émetteurs fixes sont disposés tout autour de l'utilisateur. Les récepteurs sont eux accrochés sur les parties du corps qui sont en mouvement. Les récepteurs envoient leurs mesures du champ magnétique à une unité de contrôle qui calcule la position et l'orientation de chaque traqueur.

Les traqueurs magnétiques ont pour avantage d'être petits, légers et portables, de ne pas être sensibles aux interférences acoustiques, et d'avoir un taux de rafraîchissement élevé et une faible latence. Les principaux inconvénients se situent d'abord au niveau des interférences magnétiques : des objets métalliques peuvent perturber les mesures des récepteurs. Un autre problème est lié à la technologie puisque le champ magnétique décroît rapidement avec la distance. Le volume de travail des traqueurs magnétiques reste donc assez limité. Le système le plus connu et certainement le plus utilisé est le Fastrak de la société Polhemus, dont le volume de travail – environ 5m sur 5m sur 15m – et la précision en font un excellent choix pour des environnements de type CAVE.

3.3.2 Traqueurs acoustiques

Les traqueurs acoustiques reposent sur les ondes ultrasonores (de fréquence supérieure à 20kHz) pour calculer la position et l'orientation des objets dans l'espace. L'utilisation du son ne permettant de déterminer qu'une distance relative entre deux points, il est nécessaire de disposer d'un ensemble d'émetteurs et de récepteurs simultanément. Il existe différents types de traqueurs à ultrasons : d'une part les traqueurs qui mesurent le temps d'aller-retour d'une onde, et d'autre part ceux qui comparent la phase d'une onde reçue par rapport à la phase d'une onde de référence.

Les traqueurs acoustiques sont petits et légers, relativement bon marché, et ne sont pas sujets aux interférences magnétiques. En outre ils sont assez précis. Par contre, ils souffrent d'un taux de rafraîchissement assez faible et d'interférences aux ondes acoustiques (notamment aux échos et aux bruits parasites). L'inconvénient le plus important vient du fait qu'ils sont sensibles à la visibilité : il est préférable qu'il n'y ait pas d'obstacles entre récepteurs et émetteurs. Le système Head Tracker de la société Logitech offre la possibilité de suivre les mouvements de la tête dans un volume conique 2m de long et de 100 ° d'ouverture. Le volume de travail est relativement modeste et pourra être utilisé sur un plan de travail virtuel par exemple.

3.3.3 Traqueurs optiques

Les traqueurs optiques se divisent en trois catégories. On trouve les traqueurs à base de reconnaissance de marqueurs, c'est-à-dire de formes géométriques $2D$ particulières orientées dans l'espace. Il y a également les traqueurs à marqueurs $3D$ – le plus souvent, il s'agit de boules réflechissantes : un ensemble de caméra capturent les images des marqueurs ; on peut déterminer la position et l'orientation de chaque traqueur à l'aide d'un groupe de marqueurs, puisque la

position relative des marqueurs les uns par rapport aux autres est connue et fixe. Enfin, il existe un système basé sur l'utilisation d'un transmetteur laser dont la lumière se réfléchit sur des marqueurs réfléchissants. La diffraction du laser permet de déterminer la position et l'orientation du traqueur dans l'espace.

Ces traqueurs sont totalement insensibles aux interférences acoustiques et magnétiques, la précision est relativement bonne (1mm et $0,1\,°$) et le taux de rafraîchissement très élevé (jusqu'à 250 Hz). Par contre ils souffrent de l'occlusion, de la diffraction de la lumière et des radiations infrarouges, et requièrent un calibrage minutieux. Enfin lorsque le système doit suivre plusieurs groupes de marqueurs, il peut y avoir confusion entre deux groupes. La société ARTtrack propose son système A.R.T., composé de caméras infrarouges et de marqueurs sphériques réfléchissants regroupés sur une structure rigide et portable. Chaque groupe de capteurs forme une structure tridimensionnelle que les caméras repèrent. Un traitement informatique permet ensuite de déterminer précisément la position et l'orientation de ces marqueurs par rapport aux caméras, et donc au repère global de la scène. Le volume de travail est environ de 3m sur 3m sur 3m, ce qui convient bien à une utilisation sur plan de travail ou dans un CAVE. Au delà, la précision des images perçues par les caméras n'est plus suffisante et trop distordue.

3.3.4 Traqueurs mécaniques

Les traqueurs mécaniques mesurent la position et l'orientation en connectant physiquement l'objet à suivre à un point de référence à l'aide de tiges rigides. Il s'agit de traqueurs précis qui ne sont soumis à aucun problème d'interférence. Ils sont généralement utilisés pour suivre précisément de petits volumes ; par exemple en mécanique de précision, lorsqu'on souhaite créer une copie virtuelle d'un objet physique, on peut le numériser à l'aide d'un tel dispositif.

Bien entendu, ce type de traqueur est totalement inadapté au suivi de personnes ou de gros volumes, le système étant trop intrusif puisqu'il est attaché physiquement. De plus, un espace de travail important signifie de longs liens rigides et suffisamment légers, associés à des capteurs très précis. On pourra citer le dispositif expérimental ADL-1 de la société Shooting Star Technology qui permet de suivre la tête de l'utilisateur sur 6 degrés de liberté. L'utilisateur porte un bonnet semi-rigide relié à un bras mécanique léger qui détermine la position et l'orientation. Le volume de travail de ce matériel est inférieur au m^3, ce qui constitue une sérieuse limitation aux mouvements de son porteur.

3.3.5 Traqueurs inertiels

Les traqueurs inertiels reposent sur l'utilisation d'accéléromètres et de gyroscopes. L'orientation des objets est calculée en intégrant conjointement les valeurs des gyroscopes, ces dernières étant proportionnelles à la vitesse angulaire autour de chaque axe. De la même manière, en intégrant doublement les valeurs des accéléromètres, le système retourne les variations de position.

La portée de ces capteurs est infinie. Ils sont rapides, ne souffrent d'aucun problème d'occlusion par un autre objet, ni d'interférences magnétiques, res-

tent abordables et occupent un espace très réduit. Cependant, ils ne sont pas précis lorsque les changements de positions sont lents, et restent sensibles à des phénomènes de dérives, puisque les erreurs s'accumulent en permanence. Enfin, ils ne renseignent que sur 3 degrés de liberté, ce qui est insuffisant pour suivre complètement un objet $3D$ dans l'espace. La société Systron-Donner propose son système MotionPak, un sac à dos qui contient l'ensemble du système de suivi – gyroscopes, accéléromètres – pour mesurer les mouvements linéaires et rotationnels. Il ne renseigne pas sur la position et l'orientation de l'utilisateur dans l'espace, mais uniquement sur les variations de ses déplacements.

3.4 Interfaces d'entrée utilisateur

Alors que les interfaces visuelles permettent de présenter des informations à l'utilisateur en stimulant le sens de la vision, nous allons maintenant nous intéresser à l'autre aspect de la communication homme-machine : les interfaces d'entrée utilisateur. Nous pouvons déjà considérer les traqueurs de suivi de mouvements comme des périphériques d'entrée passifs, puisqu'ils renseignent l'ordinateur sur leur propre état, mais sans fournir explicitement de commandes au système.

Les périphériques présentés dans cette section fournissent les moyens de base pour une interaction directe de l'utilisateur avec un environnement virtuel. Les commandes peuvent être entrées de bien des manières : gestes naturels de la main, sélection dans un menu, sélection d'un objet ou encore utilisation de boutons associés à des actions prédéfinies. Dans cette section nous considérons trois types de périphériques : le matériel de pointage $3D$, les gants de données et enfin les interfaces haptiques. Le choix d'une interface par rapport à une autre dépend de l'application. Pour que l'interaction devienne la plus naturelle possible, l'idéal est que chaque tâche ait son outil.

3.4.1 Périphériques de pointage $3D$

Les périphériques de pointage $3D$, comme le trackball $3D$ ou le wand, sont les principaux outils de travail que l'utilisateur manipule en réalité virtuelle. De tels dispositifs sont aussi bien employés pour des tâches de navigation, que de sélection et manipulation d'objets virtuels. Ils ne fournissent pas de retour physique à l'utilisateur, hormis celui qui est inhérent à leur forme physique. Ces périphériques reposent sur des transducteurs acoustiques, électromagnétiques, inertiels, mécaniques ou optiques, pour convertir un phénomène physique, force ou vitesse, en un signal mesurable.

Parmi les dispositifs que l'on rencontre sur le marché, il y en a peu qui diffèrent réellement en terme de fonctionnalités. La plupart du temps, il s'agit de périphériques munis d'un ou plusieurs boutons (voir figure 3.6), et associés à un traqueur pour les suivre dans l'espace. Ainsi, pour déplacer un objet, l'utilisateur attrape l'objet avec son *wand* en appuyant sur un bouton, le déplace dans l'espace à l'endroit voulu, puis relâche le bouton pour terminer le déplacement. Dernièrement, certaines firmes du jeu vidéo ont ajouté des vibreurs ou acoustiques, afin d'augmenter les sensations perçues par l'utilisateur.

Fig. 3.6: Souris $3D$ NeoWand^{TM}de Fakspace.

La société Kantek, Inc. a sorti le RingMouse qui se présente sous la forme d'un anneau que l'utilisateur glisse à son index. Sur le côté, deux boutons sont présents, et c'est le pouce qui permet de les utiliser. Un transmetteur à ultrasons permet de suivre le dispositif dans l'espace. L'avantage du RingMouse est son mode de fonctionnement sans fil et miniature. Il peut éventuellement s'utiliser en conjonction avec des gants de données (voir section suivante).

D'autres firmes choisissent une voie plus traditionnelle, comme par exemple le Flystick de la société A.R.T. ; il se présente sous la forme d'un joystick traditionnel sur lequel sont fixés des marqueurs. La différence par rapport à un joystick traditionnel est de pouvoir le suivre dans l'espace et puisqu'il est sans fil de l'utiliser librement. L'intérêt étant de conserver ses caractéristiques ergonomiques initiales pour une prise en main optimale.

La plupart des périphériques sont soit orientés vers un travail à deux dimensions – souris standard contrainte sur un plan –, soit vers une tâche tridimensionnelle. Or, il pourrait être intéressant de disposer d'une interface d'entrée qui présente de capacités d'interaction contraintes en fonction de la dimension de travail.

3.4.2 Gants de données

Les gants de données sont très intéressants dans la mesure où ils permettent d'utiliser les mains pour interagir avec le monde virtuel. La main humaine est capable de nombreux mouvements avec pas moins de 29 degrés de liberté. Chaque os de la main peut être articulé par rapport à ses voisins selon une certaine liberté qui dépend à la fois de la forme et de l'imbrication des os, et des muscles et tendons qui les relient.

On rencontre différents types de gants qui utilisent des technologies différentes. Toutes ont pour principe de mesurer des angles relatifs entre 2 parties rigides de la main. Quatre technologies sont essentiellement disponibles aujour-

d'hui sur le marché. Premièrement les exosquelettes, qui sont des gants associés à une structure semi-rigide qui entoure la main. L'idée est de mesurer les angles de flexion au niveau de la structure rigide et non au niveau des phalanges directement, les angles étant plus faciles à calculer et plus précis. Cependant, la structure de l'exosquelette est lourde et encombrante. Il est difficile de travailler librement avec ce type de dispositif. Enfin, chaque utilisateur doit porter un équipement qui convient à la morphologie de sa main.

Deuxièmement, on trouve les gants à fibres optiques. L'utilisateur enfile simplement un gant qui contient un ensemble de fibre optiques, directement placées dans le tissu. La lumière qui parcourt chaque fibre est déviée selon la courbure des doigts, et un boîtier de contrôle associé détermine les angles. Ce type de gant est très léger, et peu encombrant. Cependant, l'inconvénient majeur que l'on constate est que lorsque l'on plie un doigt, la fibre du doigt d'à côté est aussi légèrement pliée alors que ce dernier n'a pas bougé. Les mesures des angles ne sont donc pas indépendantes les unes des autres.

À mi-chemin entre l'exosquelette et la fibre optique, les gants à jauges de contrainte emploient de fines lamelles souples placées entre deux parties rigides. La pliure de chaque lamelle donne un angle, et toutes les lamelles sont indépendantes entre elles. C'est le fonctionnement du CyberGlove^{TM}de Virtual Technologies, Inc. qui retourne jusqu'à 22 angles de flexion avec une excellente précision. Avec ce système il est possible de reconnaître des gestes particuliers de la main. Par exemple, dans une application géométrique, l'utilisateur dessine un plan en posant la main à plat dans la scène, phalanges écartées, puis modifie la taille du plan en fermant plus ou moins le poing.

La dernière méthode que nous survolerons est le suivi optique des mouvements de la main et des doigts. Une ou plusieurs caméras suivent les différentes configurations de la main en détectant son contour. Le nombre de caméra doit être suffisant et leur placement bien choisi pour que toutes les postures soient visibles et détectables. Il s'agit d'une technologie délicate à mettre en œuvre qui repose notamment sur des techniques de segmentation et de reconnaissance de motifs. La société A.R.T. propose le gant Fingertracking qui offre une solution élégante de suivi optique de marqueurs fixés au bout des doigts. Certaines reconnaissances de postures échouent malheureusement, par exemple lorsque les doigts sont pliés contre la paume de la main.

Précisons que l'on peut associer à tous ces gants plusieurs options intéressantes. Le Pinch Glove de Fakespace (figure 3.7) détecte le contact entre deux doigts, ce qui est aventageux pour attraper, manipuler et relâcher un objet virtuel de façon complètement naturelle. Les gants sont généralement associés à un traqueur pour déterminer la position et l'orientation de la main dans l'espace. Finalement, cette interface est très naturelle d'utilisation et il est tout à fait envisageable de manipuler des objets virtuels avec deux ou plusieurs mains simultanément : on parle alors d'interaction bi ou multi-manuelle.

Fig. 3.7: Gant de données 5DT Data Glove^{TM}de Fifth Dimension Technologies.

3.5 Interfaces haptiques

Les interfaces haptiques permettent de stimuler le sens du toucher en interagissant physiquement avec le monde virtuel. En les utilisant, on immerge un peu plus l'utilisateur dans son environnement virtuel. L'information sensorielle haptique retournée est soit tactile, soit kinesthésique. Par exemple, lorsque l'on prend un objet en main, le sens tactile nous informe de la texture de la surface, de la géométrie de l'objet, de sa température et de la pression que nos doigts exercent. Le sens kinesthésique renseigne sur la position et le mouvement de la main et du bras, et les forces de contact totales sur l'objet.

Le retour haptique est possible de trois façons différentes. Soit activement, lorsque la machine agit physiquement sur le corps de l'utilisateur. Soit passivement, lorsque c'est le corps de l'utilisateur qui touche un objet inerte. Passive ou active, une interface physique peut devenir tangible lorsqu'elle représente physiquement l'objet virtuel que l'on manipule. Enfin, lorsque tout contact physique est impossible ou délicat à mettre en œuvre, on peut simuler visuellement le retour haptique : on parle alors de retour pseudo-haptique.

3.5.1 Interfaces haptiques actives

Les interfaces haptiques actives se composent généralement de bras mécaniques contrôlés par des servo-moteurs, qui permettent d'ajuster les forces retournées à l'utilisateur. Un ordinateur pilote l'ensemble en fonction des informations de la scène virtuelle (structure physique des objets, viscosité, pression, ...). L'interface haptique PHANToM^{TM}[MS94] de Sensable^{TM}(figure 1.3) est certainement l'exemple le plus connu. Il se présente sous la forme d'un stylo que l'utilisateur tient en main, et sur lequel s'appliquent différentes forces. On peut également citer les gants Teletact II du National Advanced Robotics Research

Centre de Salford en Angleterre dont le concept repose sur la présence de 30 poches remplies d'air sous pression, réparties sur la surface de la paume. Lorsque l'utilisateur saisit un objet, les poches se gonflent et font pression sur la peau ; il a la sensation de tenir l'objet virtuel.

Les interfaces haptiques peuvent prendre des apparences beaucoup plus imposantes, à l'image des exosquelettes dont le rôle est de contraindre les mouvements de l'utilisateur. Les exosquelettes agissent sur le sens kinesthésique en premier lieu. On trouvera par exemple le système BLEEX (Berkeley Lower Extremity Exoskeleton) de l'Université de Berkeley qui apporte une aide à la marche en analysant les influx nerveux qui dirigent les jambes. Son autonomie est très limitée et les batteries sont portées par l'exosquelette lui-même. L'université de Washington a développé l'ExoArm pour l'armée américaine, dont le but est de pouvoir supporter des charges utiles assez lourdes sur un seul bras ; Là encore, ce prototype, bien que fonctionnel, est soumis à d'énormes contraintes d'énergie et de portabilité. La figure 3.8 présente ces deux exosquelettes.

Exosquelette BLEEX

Exosquelette ExoArm

Fig. 3.8: Exosquelettes BLEEX de Berkeley (à gauche), et ExoArm de l'Université de Washington (à droite).

Les systèmes haptiques sont d'ordinaire très onéreux et complexes, à l'exception du système SPIDAR [TCH+03] qui se compose d'un traqueur attaché à quatre cordes tendues et reliées à des moteurs qui font varier la longueur des cordes dynamiquement. L'utilisateur accroche son index ou sa main au niveau du traqueur et dispose d'un retour haptique actif qui lui permet de travailler librement dans l'espace. Différentes tailles de SPIDAR sont disponibles selon l'environnement de travail ; il existe un modèle de bureau, dont le volume de travail est de l'ordre du décimètre cube, et un modèle pour plan de travail virtuel – le volume est de l'ordre du mètre cube. La présence des câbles est l'inconvénient majeur de ce système car ils ne doivent pas se croiser ou s'enrouler.

3.5.2 Interfaces haptiques passives

Les interfaces haptiques peuvent être actives comme celles que nous venons de voir, ou passives. Alors que les interfaces actives stimulent activement les sens tactiles et kinésthésiques de l'utilisateur, le fonctionnement des interfaces passives repose sur leurs propres forme et texture. Par exemple, Hinckley [HTP+97] emploie une tête de poupée que le chirurgien manipule pour simuler une opération sur un cerveau virtuel. Le dispositif est complété par une plaque en plexiglas qui sert à réaliser des vues en coupe du cerveau. De telles interfaces sont nommées des *props* [ASMR02] et donnent la sensation que l'objet virtuel (ici le cerveau) est bien réel : il a une consistance physique. Bien que les interfaces haptiques augmentent la sensation d'immersion, leur forme ne semble pas influencer les performances en phase de manipulation [WR00]. On trouvera plus d'informations dans l'état de l'art [CGL00] sur les interfaces haptiques.

3.5.3 Interfaces tangibles

Les interfaces tangibles peuvent également être considérées comme des interfaces haptiques passives. Le mot tangible vient du latin *tangere*, qui signifie toucher. De tels périphériques sont intéressants dans la mesure où ils sont assemblés comme un LEGO^{TM}pour former des périphériques destinés à une application précise. Ils sont cependant encore très limités, mais on peut imaginer construire l'équivalent physique d'une fenêtre $3D$ qui sera ensuite affichée dans le monde virtuel. L'utilisateur manipule celle-ci à la fois dans le monde réel et dans le monde physique (retour haptique passif). Ce genre d'interface n'est bien entendu pas adapté à tout. Pour manipuler un traitement de texte, le clavier reste encore la solution la plus pratique, mais pour d'autres applications comme la sculpture, le design ou encore l'architecture, une interface tangible peut s'avérer extrêmement intuitive et efficace à utiliser.

Les interfaces haptiques actives ou passives peuvent représenter un objet de la scène virtuelle. L'utilisateur dispose alors d'une sensation physique de l'objet qu'il manipule, ainsi que certaines fonctionnalités physiques qui se retrouvent dans le monde virtuel – par exemple appuyer sur un bouton physique pour manipuler un bouton virtuel. Pour [Bil01], une interface tangible est celle dans laquelle :

- chaque objet virtuel est associé à un objet physique ;
- l'utilisateur interagit avec les objets virtuels en manipulant les objets tangibles correspondants.

L'auteur donne ensuite plusieurs exemples de *widgets* basés sur des images réelles et un livre physique dont les pages contiennent des objets virtuels $3D$. Dans [GC03], le système ESKUA des auteurs est composé de 4 éléments : le cylindre, le parallélépipède, la vis, et le graft. Cette dernière pièce permet de multiplier les combinaisons possibles – elle permet d'aligner les éléments en les emboîtant. ESKUA peut être par exemple utilisé pour la CAO (conception, alignement des pièces, ...).

L'interface tangible de [PMAI03], l'Actuated Workbench, est un outil qui

utilise la force magnétique pour bouger les objets sur une table $2D$, afin de fournir un retour à l'utilisateur. Parmi les utilisations possibles, on trouve les fonctions de recherche, de tri, annuler/refaire, mettre en valeur et guider les interventions de l'utilisateur.

3.5.4 Interfaces pseudo-haptiques

L'utilisation des interfaces haptiques reste coûteuse et compliquée à mettre en œuvre. Ainsi, pour simuler des sensations haptiques sans disposer d'une telle interface, plusieurs chercheurs ont proposé d'utiliser ce qu'ils ont appelé le retour pseudo-haptique. Initialement, le retour pseudo-haptique est né de la combinaison entre un périphérique d'entrée avec un retour visuel. Par exemple, pour simuler des phénomènes de frictions, on adapte la vitesse de l'objet virtuel au support sur lequel il se trouve. Lorsque l'utilisateur dispose d'un périphérique isométrique, il appuiera plus fortement sur celui-ci pour que l'objet de la scène avance plus vite. Ce type d'interface a été beaucoup employé dans le domaine du jeu vidéo.

Lécuyer *et al.* [LCK+00] sont les pionniers dans ce domaine, en combinant un périphérique d'entrée passif avec un retour visuel. Le dispositif est utilisé pour simuler des propriétés de rigidité et de friction entre objets. Par exemple, pour simuler la friction, les auteurs proposent de réduire la vitesse de l'objet manipulé. Puisque le périphérique d'entrée est isométrique, l'utilisateur doit accroître la pression qu'il exerce sur le dispositif pour que l'objet avance. Lécuyer *et al.* [LBE04] réutilisent ce procédé pour simuler la sensation de texture de surfaces en modifiant la vitesse visuelle du curseur en fonction du micro-relief de l'objet parcouru.

Les effets pseudo-haptiques sont utilisés intuitivement dans différentes applications comme par exemple les jeux vidéos. Ainsi, lorsque le joueur marche puis nage, il devra faire plus de mouvement sur sa manette de jeu pour avancer dans l'eau que sur la terre ferme. Il a, en quelque sorte, l'impression d'être englué dans le milieu liquide. Les interfaces pseudo-haptiques n'existent pas à proprement parler ; elles n'existent que parce qu'un dispositif isométrique est associé à un retour visuel. Ainsi, du point de vue matériel, il n'y aura aucune évolution à attendre, si ce n'est peut-être dans la forme du périphérique.

3.6 Interfaces dédiées à l'ouïe, à l'odorat et au goût

Les sens liés au toucher et à la vision ont été les premiers à être mis en œuvre en réalité virtuelle. En effet, la vision constitue le sens essentiel pour représenter des données, et le toucher est celui qui est le plus important pour les manipuler. Les autres sens – l'ouïe, l'odorat et le goût – sont pourtant très importants, puisqu'ils permettent d'augmenter l'immersion. Nous allons voir que des interfaces dédiées à la stimulation de ces 3 sens sont expérimentées et donnent des résultats très intéressants.

3.6.1 Interfaces sonores

La stimulation auditive peut apporter beaucoup au réalisme d'un environnement virtuel, et pour certaines applications elle s'avère essentielle. L'utilisation conjointe de l'image et du son permet d'améliorer la précision lors de tâches de manipulation. Le son spatialisé, calculé pour prendre en compte la géométrie du monde virtuel, subit une atténuation en fonction de la distance, rebondit sur les objets, etc. L'ouïe, permet à elle toute seule, au même titre que la vue, de se positionner dans un espace en 3 dimensions. Il apparaît donc important de l'utiliser et de l'associer à des objets qui se situent tout autour de l'utilisateur. Un ensemble de haut-parleurs reproduit les actions de l'utilisateur lorsqu'il appuie sur une icône ou un bouton, mais également lorsqu'il passe à proximité d'un objet d'intérêt.

Barfield et Hendrix [BH95] ont montré que l'ajout de sons spatialisés augmente significativement la sensation de présence dans un environnement virtuel, bien qu'il n'améliore pas le réalisme apparent de cet environnement. Leur utilité en réalité virtuelle concerne également le domaine des interfaces utilisateurs ; en effet, un des problèmes rencontrés avec les interfaces graphiques modernes est l'encombrement de l'espace de travail par les icônes. Des étiquettes sonores peuvent les remplacer pour la représentation des données. Des objets et icônes audio, représentant des données, des messages, des processus, ou des ressources, peuvent être placés dans l'espace $3D$ entourant l'utilisateur, et fournir automatiquement des informations pertinentes sans pour autant qu'il y ait d'objets graphiques associés.

Certaines expériences audio virtuelles ont montré un phénomène intéressant qui apparaît lorsque le son stimule un autre organe sensoriel que l'oreille. Les ondes sonores sont véhiculées par des perturbations de l'air ou d'un milieu liquide, et dans des cas bien précis, il est possible qu'elles aient des propriétés tactiles. Par exemple, lorsque l'on ouvre une canette de soda près de notre oreille, nous entendons le son qui est associé, mais nous sentons également les bulles de dioxyde de carbone sur notre peau ; le son est produit par les bulles, et celles-ci fournissent un contact physique, même léger. Peu d'expériences ont été menées pour le moment sur ce phénomène, pourtant il représente un potentiel intéressant en environnement virtuel.

L'armée de terre américaine a développé, il y a quelques années, des armes soniques, dont le but était d'assommer ou rendre malade les ennemis. Loin de ces considérations belliqueuses, certains auteurs ont imaginé utiliser le son pour produire un retour haptique. En effet, lors d'un concert, quand la musique est forte, notre corps peut ressentir certains déplacements de l'air provoqués par les ondes sonores. Kruijff et Pander [KP05] essayent de réutiliser ce principe dans un environnement immersif, mais aboutissent à la conclusion que le système n'est véritablement efficace que lorsque le son dépasse une certaine intensité néfaste à l'oreille d'un être humain. Une solution pourrait certainement provenir des sasers[7], équivalents sonores au laser, qui amplifieraient le son en ordonnant les phonons.

[7] a. *sound amplification by stimulated emission of radiation*

Il est impossible de dresser la liste du matériel disponible pour le développement du son $3D$. En effet, avec le cinéma et les dispositifs abordables de rendu sonore à plusieurs voies – aujourd'hui 6 ou 7, la démocratisation de ces systèmes est bien réelle. Associé à des librairies de création de sons $3D$, comme OpenAL ou fmod, il est possible de créer une spatialisation sonore basée notamment sur la disposition et la nature des objets virtuels. Par exemple, dans une cathédrale, le son peut rebondir, alors que dans une chambre capitonnée, tout son sera irrémédiablement atténué.

Le domaine de l'audio virtuel a pris corps dans les industries du film, de la musique et du jeu vidéo, où les sons sont intensivement employés. Bien d'autres applications, comme la médecine, la simulation et l'architecture, ont trouvé une réelle utilité dans cette stimulation sensorielle. De nombreux progrès restent à faire dans la technologie de calcul de spatialisation et les algorithmes de rendu. En particulier, il est important de mieux contrôler le bruit ambiant réel qui distrait l'utilisateur dans son milieu immersif. Enfin, il faudra mener des recherches pour mieux comprendre l'impact des ondes sonores sur d'autres sens, en particulier l'information tactile qui pourrait aider à améliorer le retour d'informations.

3.6.2 Interfaces odorantes

La stimulation de l'odorat est l'un des derniers domaines à avoir été développé dans le cadre de l'interaction homme-machine. Il y a plusieurs raisons à cela, parmi lesquelles le manque d'applications utiles et les problèmes de technologie liés au rendu des odeurs. Pourtant, les interfaces odorantes sont aujourd'hui d'actualité. Yasuyuki Yanagi et ses collègues de l'Advanced Telecommunications Research Institute de Kyoto au Japon ont développé un canon à odeurs projettant des arômes directement dans les narines de l'utilisateur. Pour déterminer quelle odeur doit être utilisée, une caméra située sur la tête traque le mouvement des yeux. Un logiciel analyse alors les images et contrôle le flux odorant.

Pour le moment, le système a de nombreuses limitations : nombre d'odeurs restreint, portée du jet odorant, allergies potentielles aux constituants des arômes de synthèse, etc. À l'inverse des couleurs, qui peuvent être composées à partir de trois couleurs primaires, il est pour l'instant impossible de composer une odeur à partir d'odeurs essentielles. De plus, une odeur est persistante dans l'air et il faut du temps pour qu'elle disparaisse. Ainsi, le risque est de superposer plusieurs odeurs et d'obtenir un contexte odorant incohérent.

Quel est l'intérêt d'utiliser des odeurs en environnement virtuel ? Il est évident que les odeurs peuvent être employées pour modifier l'ambiance, augmenter la vigilance, diminuer le stress et augmenter l'apprentissage en les associant à des souvenirs. Certaines odeurs pourraient être vaporisées pour suggérer une ambiance positive, négative ou neutre et renforcer certaines situations. Pour le moment, les premiers dispositifs expérimentaux sont grands et trop lourds pour une utilisation prolongée, surtout lorsque des bouteilles d'air comprimé sont nécessaires. En outre, les fragrances utiles au domaine de la réalité virtuelle restent à découvrir, et cela prendra du temps et des moyens importants. Pour ces raisons, il est donc peu probable qu'un système odorant efficace soit disponible d'ici 5 à 10 ans.

3.6.3 Interfaces gustatives

Le goût est plus difficile à stimuler que l'odorat, car le périphérique de stimulation est plus invasif. En effet, bien que fonctionnant sur le même principe que le canon à odeurs, un périphérique du goût doit déposer différentes saveurs sur les papilles gustatives de la langue de l'utilisateur. Le professeur Hiroo Iwata et son équipe de l'Université Tsukuba au Japon, ont mis au point un dispositif qui simule diverses saveurs à l'aide de molécules chimiques de synthèse. Le système est complété par un dispositif sonore qui reproduit les sons de la mastication associés à l'aliment que l'utilisateur est censé manger. Pour le moment, il est possible de simuler du fromage et des biscuits secs.

Là encore, comme pour le sens de l'odorat, l'intérêt d'une stimulation gustative peut paraître limité. Il s'agit d'augmenter la sensation de présence bien sûr, mais on peut également imaginer que les interfaces gustatives viennent modifier la perception de l'environnement virtuel, en présentant à l'utilisateur plusieurs saveurs positives, neutres ou négatives. Un endroit dangereux pourrait alors être associé à un goût acide ou amer, et un lieu sûr à une saveur sucrée.

3.7 Conclusion

Pour le moment, seules les technologies liées à la vision, au suivi de mouvements et aux interfaces d'entrée sont bien adaptées pour les applications de réalité virtuelle. Pour chacune d'elles des applications commerciales existent. Les interfaces audio et haptiques restent à l'inverse cantonnées aux applications de recherche, mais sont sur le point d'être utilisées dans des applications de production. Enfin les interfaces odorantes et gustatives sont les moins étudiées de toutes les technologies présentées ici. Il est essentiel, de répondre à la question de l'utilité et de l'utilisation de telles interfaces, et du matériel utilisé. Toutes les interfaces que nous venons de voir, même les plus développées, souffrent de certaines limitations. La technologie n'exploite jamais pleinement les capacités sensorielles de l'être humain.

Dans le cas des interfaces visuelles, les casques de stéréovision et les environnements de type plan de travail ou CAVE, ont des carences bien connues : taux de rafraîchissement trop faible, champ de vision restreint, résolution faible, manque de confort, taille et encombrement qui empêchent une utilisation prolongée. Les avancées technologiques, comme l'holographie et les écrans à cristaux liquides autostéréoscopiques, et la miniaturisation joueront certainement un rôle déterminant pour la résolution de ces défauts, dès lors qu'elles seront arrivées à maturité.

Les systèmes de suivi de mouvements de la tête, des mains et du corps commencent à être bien au point. Cependant, les volumes de travail et la latence sont encore trop faibles, les technologies sont trop sensibles aux bruits, et le calibrage n'est pas automatique et trop fastidieux. Pour diminuer ces défauts, de nombreux progrès sont attendus du côté des traqueurs magnétiques et de technologies hybrides.

De nombreux dispositifs de pointage $3D$ sont disponibles ; ils représentent une technologie mature, et bien que de nouveaux produits apparaissent, il ne faut pas attendre de changements majeurs. Pour ce qui concerne les gants de données en revanche les limitations actuelles sont importantes. La morphologie des mains humaines est très complexe et différente pour chaque personne. Les technologies actuelles souffrent du manque de précision des capteurs au niveau des angles de flexion des doigts, et d'une mauvaise discrimination entre les gestes. Bien que des avancées au niveau des capteurs permettront de réduire ce problème, les algorithmes de reconnaissance de gestes joueront un rôle plus important. Là encore, des technologies hybrides pourraient améliorer la précision de la détection des angles.

Les interfaces haptiques permettent de stimuler les deux principales composantes du toucher : le retour d'effort et les sensations tactiles associées à la présentation de surfaces de contact. Pour le retour d'effort, l'utilisation d'exosquelettes et de périphériques qui opposent une résistance aux mouvements permet de contraindre les gestes de l'utilisateur. Cependant, ces systèmes sont lourds, invasifs et ne traitent que très partiellement les capacités musculaires d'un être humain. De plus, il manque des algorithmes et des modèles robustes pour la génération de signaux kinesthésiques. Pour ce qui touche au retour tactile, les défauts actuels des dispositifs haptiques concernent leurs capacités à représenter les caractéristiques des surfaces – texture, rugosité, etc. Ils sont encore incapables de représenter plusieurs sensations tactiles simultanées et le retour tactile se limite à de petites zones du corps. Les interfaces haptiques concentrent de nombreux efforts, et des avancées importantes tant au niveau logiciel que matériel peuvent être attendues d'ici les prochaines années.

Les interfaces acoustiques, bien que très utilisées dans les milieux du cinéma, de la musique, et des jeux vidéos, nécessitent de nombreuses recherches pour les adapter à une utilisation en environnement virtuel. De sérieuses limitations concernent par exemple l'ajustement du son spatialisé aux mouvements de la tête de l'utilisateur, et de la prise en compte de la nature des objets pour le rendu sonore. L'application des ondes soniques au retour tactile pourrait s'avérer efficace en combinaison avec les technologies haptiques.

Les interfaces odorantes et gustatives sont les derniers domaines sensoriels de recherche. Il n'existe pour le moment aucun système commercial, et les expériences menées restent peu nombreuses. Deux problèmes majeurs se posent à leur développement ; d'une part, l'utilité de ces dispositifs n'est pas encore bien connue et d'autre part le stockage et la libération des odeurs n'est pas au point. Il faudra en particulier déterminer quelles odeurs et saveurs de base seront intéressantes pour les environnements virtuels. Les rendus olfactif et gustatif se heurtent principalement à la persistance des odeurs et saveurs après libération ; et dans le cas de la stimulation du goût, la sensation tactile des aliments pendant la phase de mastication augmenterait sensiblement le réalisme de ce que l'utilisateur croit goûter.

Nous venons de survoler l'état actuel de la recherche et du développement des interfaces sensorielles. Leurs rôles s'avèrent déterminant pour l'immersion l'utilisateur dans un environnement virtuel. Il faudra en particulier une bien

meilleure compréhension des perceptions sensorielles humaines, de la qualité du matériel dédié et des algorithmes qui y sont combinés, avant d'obtenir un degré de stimulation réellement satisfaisant.

Deuxième partie

Réflexions et développements sur l'interaction en réalité virtuelle

Chapitre 4

La vrLib : une librairie de conception d'applications de réalité virtuelle

4.1 Introduction

Depuis les premiers travaux de Sutherland en 1965, cela fait plus de 40 ans que les chercheurs étudient les interfaces $3D$ et l'interaction homme-machine qui leur est dédiée. Depuis, force est de constater que seules quelques applications bien particulières peuvent être considérées comme abouties, stables et sont utilisées comme outil de production. Pendant des années, le coût du matériel était la principale raison avancée pour expliquer cette situation. Ce n'est plus le cas aujourd'hui : la quasi-totalité des ordinateurs actuels disposent de cartes graphiques $3D$ abordables. N'importe qui peut désormais acquérir un ordinateur dont les capacités graphiques sont équivalentes à celles des meilleures machines que l'on trouvait il y a 10 ans. Paradoxalement, la situation n'a pas vraiment changé, et il y a toujours très peu de bonnes applications $3D$. Le coût du matériel ne semble pas en être la cause.

En étudiant attentivement l'histoire des interfaces $2D$ nous pouvons trouver quelques pistes sur la nature du problème des interfaces en environnement $3D$. Nécessairement plus simples à mettre en œuvre, puisque le nombre de dimensions est moindre, il a quand même fallu attendre plus de 20 ans avant une généralisation des interfaces $2D$ sur un matériel standardisé. En effet, depuis les premiers travaux au début des années 1960, le début des années 1980 marque le début de l'ère des ordinateurs chez le particulier[1]. Ainsi, reposant sur une certaine standardisation matérielle, les développeurs ont pu ensuite pleinement se concentrer sur les aspects logiciels des interfaces homme-machine et la conception d'applications. Aujourd'hui, les librairies de techniques d'interaction fournissent un concepteur graphique d'interface utilisateur, ainsi qu'un ensemble complet de composants graphiques standards – les widgets. Un programmeur peut donc concevoir une interface utilisateur d'une application en un temps rai-

[1] a. *personal computer* ou *PC* d'IBM, et le Macintosh d'Apple

sonnable.

Bien entendu, en environnement $3D$ la situation est très différente. De nombreux périphériques d'interaction d'entrée et de sortie sont disponibles pour contrôler les interfaces utilisateur $3D$. L'environnement d'affichage peut varier énormément ; on trouve des écrans d'ordinateur, des casques de réalité virtuelle (HMD), des dispositifs de projection (comme le *Workbench*), etc. Certains permettent de suivre les mouvements de la tête de l'utilisateur, *via* les traqueurs, et ainsi recalculer un nouveau point de vue, d'autres non. Pour interagir avec ces environnements d'affichage, l'utilisateur a également le choix du périphérique d'entrée : souris $3D$, gants de données, etc. Il apparaît donc délicat de développer une application portable sur plusieurs types de configurations sans modifier le code du programme. Certaines techniques d'interaction, qui fonctionnent bien pour certains environnements, seront inutilisables sur d'autres.

Il n'existe pas de librairie standard permettant de développer des interfaces utilisateur $3D$, principalement parce qu'il existe un très grand choix de configurations matérielles [BKLP05]. Bien que de nombreux chercheurs aient proposé des techniques d'interaction dédiées à des tâches différentes [BCSH+92, LG93, PBWI96, OF03], il n'y a eu que très peu de tentatives pour offrir un véritable support pour un large panel de configurations matérielles. Le manque de librairies de développement abouties gêne considérablement la conception d'applications et de nouvelles techniques d'interaction. Une librairie standard permettrait d'offrir un cadre pour le développement de nouvelles techniques d'interaction et autoriserait la comparaison de l'efficacité des techniques entre elles. Le programmeur n'aurait donc plus besoin de réécrire entièrement une application, mais seulement changer d'outil d'interaction en fonction de l'environnement. De plus, une telle librairie pourrait offrir des techniques de base qu'elle intègre en standard, comme par exemple celle du lancer de rayon.

Le but fixé par la vrLib est de fournir un ensemble cohérent, simple d'utilisation et efficace, d'outils pour le développement des applications de réalité virtuelle, en simplifiant la conception, le développement et l'utilisation des outils d'interaction et des interfaces graphiques. Nous souhaitons proposer à la fois des objets et des outils d'interaction qui permettent de transposer le plus simplement possible des applications $2D$ en environnement $3D$, puis d'améliorer l'interaction et l'interface. Ceci sous-entend que les programmeurs ne souhaitent pas avoir à réinventer la roue à chaque application en écrivant du code déjà disponible, testé et éprouvé. Ce point est particulièrement vrai pour les outils d'interaction : en effet, de nombreuses applications contiennent l'implantation de la métaphore du lancer de rayon, bien qu'elle soit très simple et très utilisée dans le domaine.

4.2 Motivations

L'objectif de cette section est de donner un aperçu de l'existant en matière d'outils de développement spécifiques aux environnements de réalité virtuelle. Dans un premier temps, dressons un bref état de l'art sur deux classes de boîtes à outils qui nous intéressent dans le cadre de notre travail. Les premières, dites

de bas-niveau, qui sont essentiellement dédiées à la gestion et à l'abstraction du matériel et à la production d'images stéréoscopiques. Les secondes, dites de haut-niveau, qui sont focalisées plus spécifiquement à la gestion des tâches d'interaction et à la création d'interfaces graphiques utilisateur pour les environnements virtuels. Nous omettons volontairement les librairies de calculs et de simulation telles que OpenMask [Ope], de communications et couplage de codes parallèles, qui sortent du cadre de notre travail.

Dans un second temps, nous donnons les motivations qui ont conduit à la réalisation de ce travail de thèse, en indiquant la philosophie que nous avons choisi de mettre en œuvre, à travers, entre autre, la vrLib.

4.2.1 État de l'art

On trouve de très nombreuses librairies consacrées, de près ou de loin, au domaine de la réalité virtuelle. Nous estimons qu'il en existe environ une cinquantaine actuellement, ce qui montre le besoin d'avoir un certain nombre d'outils et de mécanismes de programmation pour développer des applications en environnement immersif.

La concomitance de toutes ces librairies montre plusieurs aspects. Premièrement, la librairie idéale n'existe pas, beaucoup de développeurs choisissent de développer leur propre boîte à outils. Deuxièmement, chaque librairie essaye de tirer parti d'un nombre limité de périphériques physiques et de configurations matérielles. Enfin, il n'y a aucun standard de développement, les techniques d'interaction ne sont pas clairement définies, comme peut l'être le pointeur de souris dans un environnement $2D$.

Cette section nous permet de couvrir un bref état de l'art sur les librairies les plus intéressantes dédiées aux techniques d'interaction en réalité virtuelle, et plus généralement au développement d'interfaces utilisateur. Nous choisissons volontairement de nous restreindre à celles qui nous ont semblé les plus pertinentes. Le lecteur trouvera d'abord une section consacrée aux librairies de bas niveau, plus proches du matériel sous-jacent, puis celles de haut-niveau, plus proches du programmeur d'interaction.

Librairies de bas niveau

Les librairies de bas niveau sont essentiellement consacrées à traiter et abstraire la couche matérielle pour le programmeur. Pour le domaine de la réalité virtuelle, les configurations stéréoscopiques à plusieurs contextes partagés sur des ordinateurs différents ne sont pas rares. Les boîtes à outils que nous allons décrire proposent la plupart du temps des mécanismes de gestion de la mémoire et de synchronisation des contextes graphiques (threads, mutex, ...). Voici les 3 librairies que nous avons testées.

CAVElib [VRC] est une librairie commerciale qui permet d'intégrer des programmes OpenGL, Performer et Inventor dans un environnement immersif de réalité virtuelle. Développée en langage C, elle fournit des mécanismes d'affichage stéréoscopiques et de suivi du mouvement sous plusieurs plateformes

différentes (Windows, Linux et Irix). Les fonctions de CAVElib maintiennent tous les dispositifs matériels synchronisés, produisent la perspective correcte de chaque écran, maintiennent les capteurs de position et d'orientation en service, et fournissent aux applications l'état actuel de tous les éléments du système (wand, traqueurs, ...). L'initialisation de l'environnement est réalisée une fois pour toute ; c'est notamment durant cette phase que le processus de rendu est lancé, et que les variables liées aux traqueurs sont initialisées. Ensuite, les informations renvoyées par les capteurs sont transmises en mémoire partagée *via* le programme trackd. Ceci permet à chaque tâche (calculs, entrées, rendu) de fonctionner à sa propre fréquence, mais alourdi le développement de l'application. CAVElib est encore très utilisée aujourd'hui, surtout parce que la société qui l'édite propose un support conséquent, mais cette librairie tend doucement à être remplacée par VRJuggler.

FreeVR [Fre] est une librairie qui fournit une couche d'abstraction pour les périphériques d'entrée et de sortie uniquement. Cette boîte à outils libre est réalisée par Bill Sherman. Elle permet de produire en environnement virtuel stéréoscopique quelle que soit la configuration des écrans de projection. FreeVR ne fournit cependant aucun mécanisme relatif à la gestion des entités virtuelles (graphe de scène, outils d'interaction, ...). Elle prend en charge l'abstraction des périphériques d'entrée et de sortie bien que leur description soit plus complexe qu'avec une librairie telle que VRJuggler, et dispose également d'un mode simulation pour tester une application de réalité virtuelle développée sur station de travail.

VRJuggler [VRJ] est une librairie libre de bas et haut niveau développée à l'Université de l'état d'Iowa par l'équipe de Carolina Cruz-Neira. Elle permet de concevoir des applications de réalité virtuelle supportant de nombreuses plateformes (stations SGI, PC sous Linux ou Windows, Mac). Carolina Cruz-Neira a initié ce projet dans le but de fournir un cadre d'application plus mature et surtout libre, permettant le partage des ressources développées dans la communauté. VRJuggler permet à l'application de faire abstraction des dispositifs matériels utilisés, ce qui facilite l'écriture des programmes quels que soient les périphériques utilisées.

Des pilotes existent pour un grand nombre de dispositifs, et cet environnement offre une grand liberté quand à l'architecture de l'application et la méthode d'affichage, tout en prenant en charge les fonctionnalités de calcul du point de vue stéréoscopique à partir des traqueurs disponibles et de la disposition des différentes surfaces d'affichages.

En interne, le fonctionnement de VRJuggler s'articule autour d'un micronoyau qui contrôle la boucle d'exécution de l'application en faisant appel à un ensemble de gestionnaires. Le gestionnaire *ConfigManager* gère la configuration de l'application, ainsi que les possibles re-configurations en cours d'exécution. Le gestionnaire *PerformanceManager* prend en charge le suivi de l'exécution en vue d'obtenir des statistiques de performances. Le gestionnaire *InputManager* sert à contrôler les périphériques d'entrées à l'aide de plugins qui sont chargés en mémoire. Le gestionnaire *DisplayManager* gère l'affichage quelle que soit l'API sous-jacente – OpenGL, Performer, OpenSG et OpenSceneGraph – et quelle que soit la plateforme ; il calcule en particulier les différentes matrices de

projections. Enfin, le gestionnaire *DrawManager* prend en charge le rendu en fonction de l'API utilisée.

VRJuggler contient également des outils graphiques permettant d'éditer la configuration de l'application, ainsi qu'un mode simulation permettant de tester les applications conçues pour des environnements immersifs sur station de travail. Très utilisée aujourd'hui, elle propose à la fois l'accès direct au matériel (bas niveau), et des mécanismes d'interaction de plus haut niveau. Ses points forts sont ses mécanismes de gestion multi-threads des processus graphiques, sa couche d'abstraction matérielle, et son extension ClusterJuggler qui autorise l'utilisation de grappes de PC lorsque de gros calculs sont nécessaires. C'est la librairie que nous utilisons actuellement dans notre équipe de recherche.

Comme nous venons de le voir, plusieurs librairies sont en concurrence sur l'abstraction matérielle et la gestion de la stéréoscopie. Dans notre laboratoire, nous avons successivement testé CAVElib, FreeVR et VRJuggler. Finalement, notre choix s'est arrêté sur VRJuggler, puisque CAVElib est commerciale, n'évolue plus vraiment, et rencontre des limites pour des configurations matérielles particulières. FreeVR ne supporte qu'un nombre limité de périphériques, et offre moins de souplesse au niveau de sa configuration que VRJuggler, qui est libre de droit et repose sur un processus de développement communautaire libre.

Librairies de haut niveau

Les librairies de haut niveau proposent des outils de programmation permettant de développer assez facilement une application de réalité virtuelle. Elles permettent par exemple de créer une interface graphique et des outils d'interaction qui peuvent manipuler cette interface. De nombreuses tentatives ont déjà été réalisées sans pour autant aboutir à une réponse très satisfaisante. Les librairies NICElib et libSelect [SBB03] constituent nos deux précédentes tentatives d'obtenir une librairie d'interaction dédiées à l'écriture d'applications en environnement $3D$. NICElib repose de la classification des techniques d'interaction pour proposer des métaphores de navigation, de sélection et de manipulation qui peuvent être combinées entre elles selon les tâches à réaliser. Ce système, trop rigide, et contraignant en terme de programmation, nous a permis d'expérimenter les techniques d'interaction les plus connues (lancer de rayon, GoGo, C^3, ...). À l'inverse, notre seconde librairie, la libSelect, est plus orientée sur la production d'une interface graphique dans un environnement $3D$. Ces deux boîtes à outils restent très expérimentales et trop rigides pour pouvoir développer facilement de nouveaux outils d'interaction. Nous allons présenter les boîtes à outils qui nous ont semblé être les plus pertinentes par rapport à notre approche de l'interaction.

3DWM [3DW] est une boîte à outils libre réalisée par Niklas Elmqvist de l'Université de Göteborg ; elle est basée sur une architecture type graphe de scène permettant de construire des interfaces utilisateurs $3D$. Les composants graphiques $2D$, comme les widgets $2D$, sont affichés comme des textures sur des objets $3D$ sur lesquelles on agit à l'aide d'un lancer de rayon virtuel. Concrètement, les auteurs ont développé un serveur graphique sur le modèle d'un serveur X, auquel une application cliente peut se connecter et, d'une part, demander

l'affichage d'objets $3D$, et d'autre part, recevoir des événements tels qu'un clic, afin de permettre ne interaction avec l'utilisateur.

L'architecture de 3DWM se rapproche de celle d'un serveur X ; écrite en C++, elle emploie CORBA qui sert d'interface entre les différents composants du système et les applications externes. La plateforme est elle-même divisée en différents composants indépendants. L'interface avec le système permet de connecter les différents composants entre eux, et sert de point d'entrée à toute application externe. Le gestionnaire d'évènements fait tourner la boucle principale du système. Le module est conçu de manière à éviter toute attente active, et fournit l'ensemble des fonctionnalités à une programmation événementielle. La base de données de la scène sert au stockage de la hiérarchie des informations concernant la scène à représenter, ainsi que les fonctions de manipulation associées. Le gestionnaire d'entrées traduit les événements retournés par les dispositifs d'entrée en commandes pour le système. Enfin le gestionnaire de plugins sert de base à l'écriture de modules d'extensions pour de nouvelles techniques d'interaction.

L'implantation de ce projet s'est essentiellement focalisée sur la technique du $3D$ Cube ; il s'agit d'un cube dont chacune des faces affiche un ensemble de widgets $2D$. Ainsi, plutôt que de créer une interface graphique purement $3D$, le projet 3DWM propose de réutiliser les widgets $2D$ qui sont bien connus en leur trouvant de nouveaux supports d'expression. Bien que ce projet C++ et multiplateforme soit abandonné depuis fin 2003, nous le citons pour ses capacités d'interaction multi-dimensionnelle ($1D$, $2D$ et $3D$).

Grappl $3D$ [GL04] est une librairie d'interaction $3D$ parmi les plus récentes de toutes celles qui existent. Elle est issue des travaux de Mark Green sur MR ToolKit [Too] de l'Université d'Alberta. Elle permet de produire des interfaces utilisateur $3D$ qui s'adaptent à la configuration matérielle sur laquelle tourne l'application au moment de l'exécution. Ainsi, au démarrage du programme, le système détermine quels sont les périphériques d'entrée et de sortie disponibles, puis sélectionne les techniques d'interaction les plus appropriées en modifiant éventuellement l'interface utilisateur. Il s'agit d'une idée très intéressante, car cette librairie simplifie le développement d'application $3D$. Cependant, le contrôle précis de l'interaction et de l'interface se revèle assez délicat. Les auteurs ont ajouté une fonctionnalité de placement automatique des éléments graphiques qui constituent l'interface $3D$ grâce au mécanisme des *layout* $3D$. Pour le moment, il est difficile de savoir ce que permet exactement cette librairie puisqu'il existe seulement trois articles sur les choix architecturaux faits par leur auteur principal Joe Lo. Il est donc impossible de la tester convenablement.

vtkVR [vtk] est la dernière librairie que nous citerons ici. Elle a été conçue par M. Scarpa et R.G. Bellemande de l'Université d'Amsterdam. Cette boîte à outils fournit différentes couches d'asbtraction qui viennent s'imbriquer les unes dans les autres, et qui sont configurées à l'aide de plusieurs fichiers de configuration. La couche d'abstraction aux périphériques de sortie permet par exemple de s'abstraire des spécificités matérielles de l'affichage. La couche d'indépendance aux périphériques d'entrée fournit un ensemble de 4 sous-couches qui servent à écrire une application, et contient certaines techniques d'interaction de base pour la sélection et la manipulation des objets virtuels.

La première couche "Application" constitue le niveau le plus abstrait, et le programmeur spécifie des choses du genre "Je veux que l'utilisateur soit capable de sélectionner un de ces objets". La seconde couche "Technique d'Interaction", permet d'indiquer quels outils doivent être employés pour telle tâche d'interaction. La troisième couche "Correspondance aux entrées", sert à indiquer que telle technique d'interaction nécessite tels types d'entrées. Enfin, la quatrième et dernière couche "Abstraction du matériel", permet d'abstraire les dispositifs d'entrée à l'aide de représentations génériques qui seront utilisées par la seconde et troisième couche selon les besoins d'une technique d'interaction.

Pour l'instant, vtkVR est en phase de développement, et ne fonctionne que sur environnement Linux en ne supportant qu'un nombre limité de périphériques et de techniques d'interaction. Cependant, son modèle d'abstraction du matériel d'entrée et de sorties, permet d'écrire une application plus générique.

À l'inverse des librairies de bas niveau qui se focalisent sensiblement toutes à abstraire le matériel physique, les boîtes à outils de haut niveau proposent des solutions de développement très hétérogènes. Grappl $3D$ est assez prometteur, tout au moins en théorie puisqu'aucun exemple n'est visible. Il offre une bonne gestion du matériel et permet d'écrire des techniques d'interaction assez simples. Sur ce point, cette librairie se rapproche sensiblement de nos précédents travaux sur la NICElib et la libSelect. Le mécanisme d'abstraction du matériel mis en œuvre dans vtkVR nous semble très intéressant dans la mesure où il permet d'écrire des techniques d'interaction évoluées très simplement. Les librairies de programmation d'interfaces graphiques $3D$ sont rares. La solution proposée par 3DWM est un premier pas en avant pour faire le lien entre les interfaces en environnements $2D$ bien connus, en environnement $3D$. Finalement, à notre connaissance il n'existe aucune librairie qui propose à la fois un ensemble d'outils d'interaction indépendants du matériel, et des widgets de conception d'une interface graphique, dédiée à la conception d'une application de réalité virtuelle.

4.2.2 Motivation et philosophie

Nous venons de voir que de nombreuses librairies offrent un cadre de développement pour les environnements de réalité virtuelle, mais pour des domaines bien ciblés. Dune part, il existe celles dites de bas-niveau et dont le rôle est de gérer le matériel des périphériques d'entrée et de sortie, ainsi que la production d'images stéréoscopiques, mais qui ne proposent aucune technique d'interaction. D'autre part, on trouve celles dites de haut-niveau, qui s'appuient sur une librairie de bas-niveau pour la gestion du matériel et de l'affichage et qui proposent des outils d'interaction et/ou des widgets dédiées à la conception d'une interface graphique pour une application.

La pluralité des librairies, leur interdépendances, et leur liens très étroits avec le matériel les rendent relativement compliquées à utiliser lors de la réalisation d'applications de réalité virtuelle qui nécessitent une interface graphique et des outils d'interaction adaptés et simples à mettre en œuvre. En particulier, aucune ne permet de porter facilement une application d'un environnement $2D$ – de type station de travail avec clavier et souris, à un environnement $3D$ – de type *Workbench* avec wand et gants de données.

Le manque d'une plateforme qui proposerait à la fois des outils d'interaction indépendants du matériel sous-jacent, et un ensemble widgets nécessaires à la conception d'une interface graphique utilisateur, a pour conséquence de baisser la production du programmeur. Une des causes à cet état de fait est à chercher dans le clivage logiciel qui existe depuis toujours entre les environnements $2D$ et les environnements $3D$. On trouve de chaque côté des techniques d'interaction et des widgets dédiés à l'environnement en question. Nous souhaitons au contraire que l'interaction $2D$ et $3D$ soient réunies dans un tout cohérent, de manière à pouvoir utiliser en environnement $3D$ des outils et des widgets issus directement des environnements $2D$ et qui sont bien connus et éprouvés. Ce n'est pas parce qu'une application est dite de réalité virtuelle qu'elle ne doit pas proposer des outils $1D$ ou $2D$, comme par exemple le pointeur de souris. Ceux-ci ont fait leurs preuves et ne sont pas nécessairement obsolètes en environnement $3D$.

La vrLib n'a pas la prétention de révolutionner la programmation d'application de réalité virtuelle, et encore moins l'interaction. Au contraire, elle doit avant tout offrir un cadre de développement bien connu de n'importe quel programmeur, qu'il soit expert ou non en réalité virtuelle. Le code écrit doit s'apparenter autant que possible à celui des outils existants en $2D$ – Qt de Trolltech, Swing Java de SUN, Visual Studio de Microsoft, GTK+, etc. Il y a à ce niveau, comme nous venons de le voir, un véritable manque d'outils de développement.

Parmi les logiciels développés pour environnement immersif, on trouve une grande part de programmes qui sont portés depuis un environnement de bureau. Le développeur doit alors repenser entièrement l'interaction à partir d'une feuille blanche, puisqu'il ne dispose que de peu d'outils pour la $3D$. La vrLib doit faciliter le portage de telles applications, afin qu'elles soient fonctionnelles rapidement, même si l'interaction n'est pas optimale dans un premier temps. Le programmeur ne doit pas perdre de temps sur la réécriture de l'interface et de l'interaction. En particulier, il ne doit jamais avoir à prendre en compte le matériel sur lequel doit faire fonctionner le programme.

Dans la suite de cette section, nous parlons d'applications $2D$ et $3D$. Nous proposons une définition simple de ces termes.

Application $2D$ Une application $2D$ est un programme dont l'interface graphique tourne sur environnement $2D$, de type station de travail, et se composent traditionnellement d'objets graphiques de type Fenêtres, Boutons, et Menus déroulants $2D$.

Application $3D$ On parle d'application $3D$ par opposition à application $2D$, pour indiquer que l'interface graphique de l'utilisateur est prévue pour fonctionner sur environnement $3D$ de type *Workbench*.

Lors de l'écriture d'une application, la vrLib doit être en mesure d'aider le développeur à plusieurs niveaux. Dans le cas où celui-ci doit porter une application $2D$ existante sur station, la librairie devra offrir des outils d'interaction et des widgets directement issus de la $2D$. Puis, le développeur pourra faire progressivement évoluer l'interface et les outils d'interaction en les remplaçant

par de nouvelles techniques plus adaptées et bien pensées. Dans le cas où le développeur doit écrire une application entièrement en environnement virtuel, la vrLib doit apporter un minimum d'outils d'interaction et de widgets $3D$.

En tant que concepteurs de solutions logicielles pour la réalité virtuelle, nous constatons que le matériel est très expérimental et ralentit le développement lorsqu'on travaille directement dessus. Dans un premier temps, il vaut mieux programmer sur station, autant que possible, puis compléter l'interface et l'interaction en environnement immersif. Un simple fichier de configuration devra servir à décrire l'interface et les outils d'interaction et permettra, sans reprendre le code déjà écrit au sein de l'application, de passer d'une station de travail à un environnement de réalité virtuelle.

La philosophie de la vrLib repose donc sur deux concepts : d'une part, nous souhaitons fournir au programmeur des outils similaires à ceux dont il a l'habitude en environnement $2D$, de façon à ce qu'il puisse développer rapidement et facilement une application de réalité virtuelle. D'autre part, la librairie doit également lui permettre d'utiliser et de rajouter, avec beaucoup de flexibilité, des techniques d'interaction appropriées à la réalité virtuelle et aux environnements $3D$. En particulier, il est indispensable que l'utilisateur de cette application retrouve autant que possible les automatismes qui sont les siens en $2D$. Ces deux points constituent la puissance de la vrLib.

4.3 Architecture

Cette section couvre l'architecture mise en œuvre dans la vrLib. Nous décrivons dans une première partie la structure globale de la librairie, en donnant le déroulement d'une application point par point *via* la boucle événementielle de la vrLib. Dans un deuxième temps, nous expliquons les mécanismes de messages systèmes et de signaux propres à la gestion des événements. Nous décrivons ensuite la hiérarchie des objets de la librairie au sein de son graphe de scène, ainsi que les objets les plus importants proposés au développeur. Une quatrième partie est consacrée au fonctionnement des outils d'interaction et à l'interaction adaptative. Enfin, une courte et dernière sous-section décrit les contraintes logicielles et matérielles rencontrées lors du développement de la vrLib. La figure 4.1 donne l'imbrication de la vrLib et de ses constituants avec le programme qui l'utilise et le système d'exploitation sur lequel elle repose.

4.3.1 Structure globale de la vrLib

Le déroulement globale d'une application écrite avec la vrLib est schématisé par la figure 4.2. Rappelons que la vrLib a pour but de permettre la description d'une scène de réalité virtuelle par la représentation géométrique de ses objets. Ceux-ci sont placés dans une hiérarchie et manipulés à l'aide d'outils d'interaction. La librairie est construite autour du mécanisme de passages de messages entre les outils d'interaction et les objets de l'application, ou directement entre un ou plusieurs objets. Lorsqu'aucun objet n'est sélectionné par un outil d'interaction, c'est l'objet univers ou racine qui reçoit les messages générés.

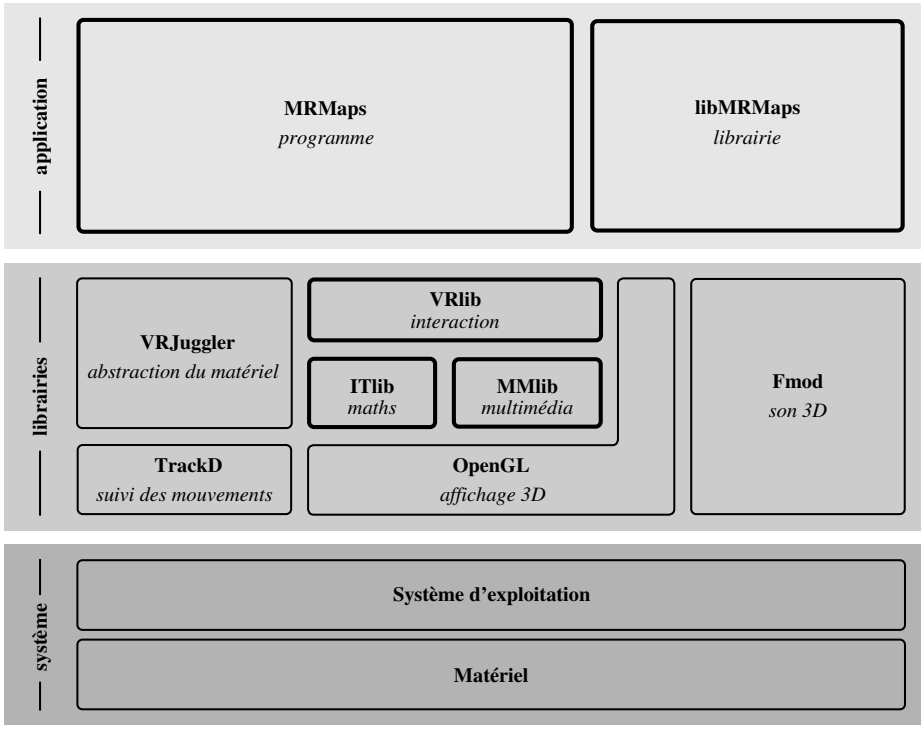


Fig. 4.1: Architecture de la vrLib.

Techniquement, le déroulement d'une application comporte essentiellement trois phases distinctes. La première correspond à la création de l'objet application, à son initialisation et à l'initialisation des données dépendantes du contexte graphique ; cette phase n'est exécutée qu'une seule fois, au lancement de l'application. La seconde correspond à la boucle de gestion des événements et de l'affichage des objets et outils d'interaction de la librairie. Enfin la troisième et dernière phase correspond à la destruction de l'application, de ses objets, et du nettoyage de la mémoire. Nous pouvons en donner un déroulement point par point :

1. construction de l'objet application ;
2. initialisation de l'application avant que l'API graphique soit démarrée ;
3. initialisation des paramètres de l'application dépendant du contexte graphique, après que l'API graphique soit démarrée, mais avant que la boucle d'événements et de rendu soit lancée ;
4. traitements nécessaires à chaque itération de la boucle événementielle, avant l'affichage. À ce niveau, nous rafraîchissons les données relatives au matériel, puis les messages sont générés et traités, et enfin, tous les objets qui sont en cours de destruction[2] sont supprimés de la liste des outils

[2]Lorsque l'utilisateur souhaite détruire un objet, la librairie se charge de détruire en interne toutes ses références, afin que l'application ne se termine pas sur une erreur.

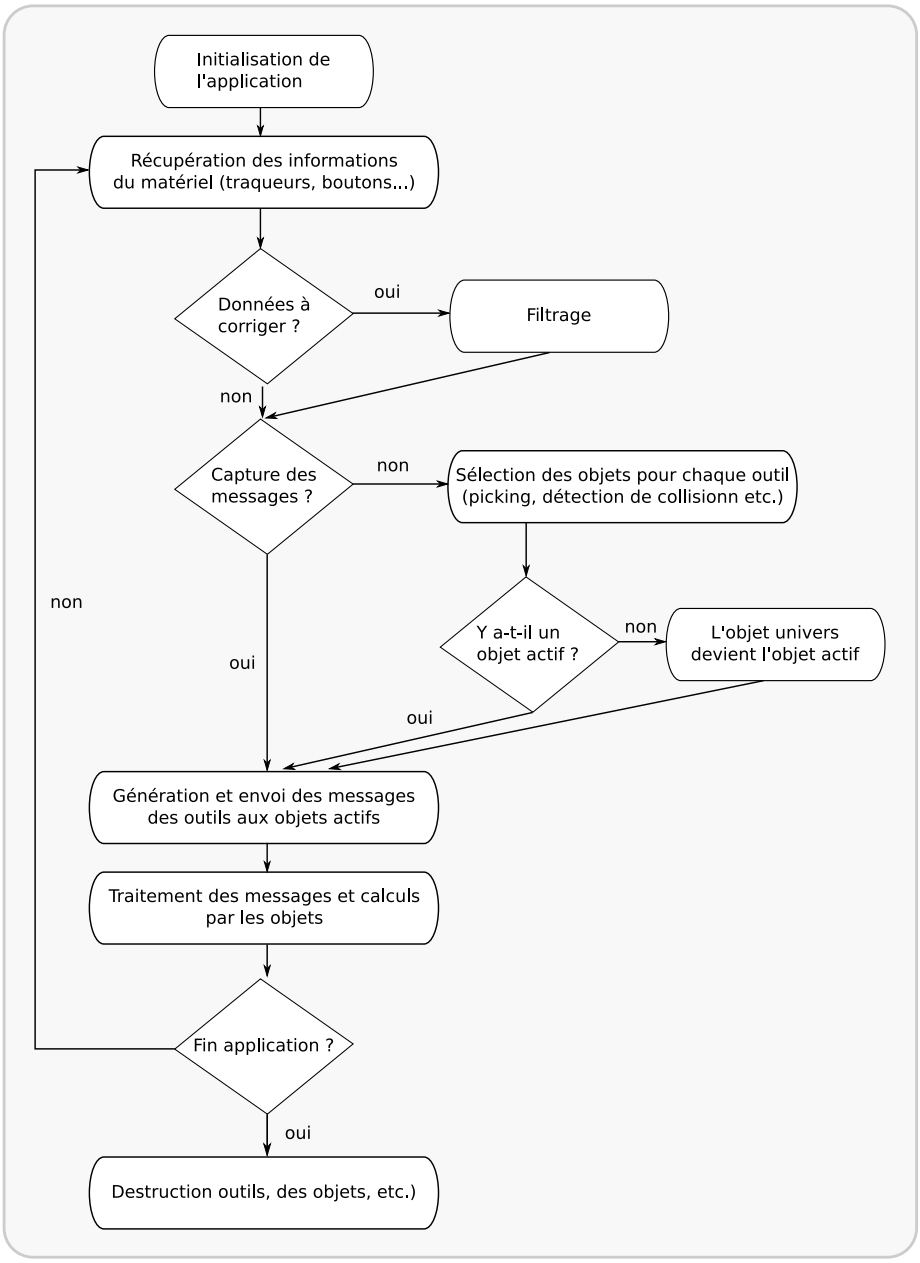


Fig. 4.2: Déroulement d'une application vrLib.

d'interaction. Ce dernier traitement évite de continuer à manipuler des objets qui n'existent plus et de terminer l'application ;
5. affichage de l'arborescence des objets de la librairie. À ce niveau, toute la hiérarchie des objets de l'application qui sont pris en charge par la vrLib sont affichés, ainsi que les outils d'interaction que l'utilisateur manipule ;
6. traitements nécessaires à chaque itération de la boucle événementielle, après l'affichage. Ici, il est possible de rajouter du code dépendant du contexte graphique, mais après que l'affichage ait eu lieu ;
7. fin de la boucle des événements, et fin de l'application.

4.3.2 Gestion des événements

La librairie vrLib utilise intensivement deux mécanismes de communication entre la librairie, l'application et les objets, selon les événements du système et de l'utilisateur. Les messages systèmes, qu'ils soient de bas-niveau ou de haut-niveau, sont retournés par la librairie à l'application et aux objets, à l'aide d'une valeur qui représente le type d'événement. Ces messages contiennent également l'identifiant de l'envoyeur, afin de permettre aux objets de réaliser une action spécifique. Les signaux et slots sont dédiés à la communication entre objets de l'application. Comme pour les messages systèmes, ils contiennent l'objet qui a émis le signal. Lorsqu'un objet reçoit un message ou signal, il peut réagir d'une manière prédéfinie et totalement adaptée, en fonction des souhaits du programmeur. Nous allons détailler les notions de messages systèmes et signaux/slots.

Messages systèmes

Définition Un message système, au sens de la vrLib, est un message qui est envoyé par l'application aux objets virtuels pour leur indiquer une modification de l'état d'interaction.

Chaque périphérique physique, comme par exemple un gant de données, un joystick ou encore un clavier, produit une ou plusieurs informations matérielles, et se compose d'un ensemble d'éléments. Un élément est une pièce du périphérique qui génère des événements de même type. Par exemple, un élément déclencheur, tel qu'un bouton, peut générer des événements déclencheurs. Ces événements communiquent leurs changements d'état à la librairie qui a la charge de distribué les messages, soit en les envoyant sous forme de messages de bas niveau, soit en les traitant de façon plus complexe et en les envoyant *via* des messages de haut niveau.

Messages de bas niveau Chaque message de bas niveau est envoyé par un outil d'interaction à un ou plusieurs objets lorsque le volume de l'outil entre en contact avec le ou les objets, et qu'un événement physique se produit sur le périphérique physique associé à la métaphore. Il s'agit du mécanisme de la vrLib pour abstraire les dispositifs matériels.

Ces messages sont répartis en quatre groupes : les déclencheurs – événement enfoncé / relâché, les valuateurs – événement continu, les traqueurs – changement de position et d'orientation – et les claviers – manipulation d'une ou

plusieurs touches. Chaque message est envoyé avec un identifiant unique qui sert à le retrouver.

Message Déclencheur Un déclencheur[3] est un élément qui se trouve soit dans l'état "allumé/enfoncé" ou "éteint/relâché". Un message déclencheur indique une transition d'un état à un autre. Les déclencheurs représentent souvent des périphériques physiques comme les boutons et les interrupteurs. Chaque message déclencheur contient l'état courant, et l'état précédent ; de cette manière, il est également possible de dire si l'état est en cours de changement ou non.

Ainsi, chaque message déclencheur propose :

- son état courant ("enfoncé" ou "relâché") ;
- son état précédent ("enfoncé" ou "relâché") ;
- son changement d'état ("aucun", "en cours d'enfoncement" ou "en cours de relâchement") ;
- un identifiant vers l'outil d'interaction qui a généré le message.

Le message Déclencheur est envoyé aux objets virtuels pour leur indiquer qu'il puissent définir un certain comportement lorsque l'utilisateur enfonce un bouton du périphérique qu'il manipule.

Message Valuateur Un valuateur est un élément auquel est associée une valeur réelle. Cette valeur peut être bornée, c'est-à-dire associée avec une plage de valeurs possibles déterminée par un minimum et un maximum, ou au contraire non bornée, c'est-à-dire que la plage de valeur est limitée aux capacités de la variable de stockage. Les valuateurs représentent typiquement des dispositifs physiques qui produisent des valeurs continues – ou pseudo continues, comme par exemple l'angle de flexion d'une phalange.

Ainsi, chaque message valuateur propose :

- sa valeur courante ;
- sa valeur prédédente ;
- un identifiant vers l'outil d'interaction qui a généré le message.

Message Traqueur Un traqueur est un périphérique d'entrée dont le rôle est de retourner sa propre position et orientation dans l'espace. À chaque message Traqueur est associée une matrice de transformation 4x4 en coordonnées homogènes donnant la position et l'orientation actuelle et précédente du traqueur considéré, comme par exemple la tête de l'utilisateur. En conservant les informations précédentes, le traqueur permet également de déterminer son déplacement et ses changements d'orientation.

Ainsi, chaque message traqueur propose :

[3] a. *trigger*

- sa transformation courante ;
- sa transformation prédédente ;
- un identifiant vers l'outil d'interaction qui a généré le message.

Le message Traqueur peut être utilisé pour déplacer un objet virtuel. Par exemple, avec la technique du lancer de rayon, ce message sera envoyé à l'objet pour lui indiquer sa nouvelle position et orientation en fonction du périphérique que l'utilisateur tient en main.

Message Clavier Un événement clavier intervient lorsque une touche d'un clavier est enfoncée. Il s'agit en fait d'un ensemble de déclencheurs réunis pour composer le clavier. À tout moment, il est ainsi possible de connaître l'état courant et précédent de chaque touche du clavier – "enfoncée" ou "relâchée", ainsi qu'un éventuel changement d'état. Un message clavier est envoyé lorsqu'un changement d'état se produit sur le périphérique physique qui est associé. Précisons qu'un message Clavier contient uniquement l'état des touches qui sont enfoncées, ou qui viennent d'être relâchée, ce qui évite d'associer à chaque message l'état de toutes les touches.

Ainsi, chaque message clavier propose :

- l'état courant de chaque touche ("enfoncé" ou "relâché") ;
- l'état précédent de chaque touche ("enfoncé" ou "relâché") ;
- le changement d'état de la ou des touches ("aucun", "en cours d'enfoncement" ou "en cours de relâchement") ;
- un identifiant vers l'outil d'interaction qui a généré le message.

Ce message peut être associé à un véritable clavier de station de travail, ou encore à un système de reconnaissance, comme le T9 sur téléphone portable. Il devient ainsi possible de rentrer des données textuelles à l'application pour, par exemple, sauvegarder un fichier sous un certain nom.

Messages de haut niveau Alors que les messages de bas niveau correspondent, dans une certaine mesure, aux informations des éléments d'un périphérique particulier, les messages de haut niveau se situent plus spécifiquement au niveau de l'outil d'interaction qui les envoie.

La librairie peut envoyer quatre messages aux différents objets de l'application : les message "entrer" lorsque le volume de l'outil pénètre dans celui de l'objet, le message "sortir" lorsqu'il le quitte, le message "déplacer" lorsque l'outil bouge dans ou avec l'objet, et enfin le message "déposer" lorsque l'outil amène un objet sur un autre. Nous verrons, dans la section qui traite des objets et de leur hiérarchie, que les messages de haut niveau permettent de faire la distinction entre plusieurs zones différentes de l'objet. Par défaut, la détection du contact entre les outils d'interaction et les objets virtuels est laissée à la charge d'OpenGL, *via* le mécanisme du picking. Cette information est ensuite retraitée et enrichie pour chaque couple outil - objet, de manière à déterminer comment l'outil agit sur l'objet – entre-t-il en contact avec l'objet ? en sort-il ? se déplace-t-il à sa surface ?.

Message Entrer Lorsqu'un outil d'interaction entre dans un objet, il envoie un message "Entrer" qui permet de savoir à quel endroit sur l'objet, l'outil est entré en contact. Ce message correspond à l'événement `onMouseEnter` dans de nombreuses librairies $2D$.

Ainsi, chaque message "Entrer" propose :

- le repère local correspondant à l'intersection entre l'objet et l'outil ;
- un identifiant vers l'outil d'interaction qui a généré le message ;
- un identifiant unique.

Message Sortir À la manière du message "Entrer", le message "Sortir" indique que l'outil d'interaction quitte l'objet avec lequel il était en contact jusque là. Ce message correspond à l'événement `onMouseLeave` dans de nombreuses librairies $2D$.

Ainsi, chaque message "Sortir" propose :

- le repère local correspondant à l'intersection entre l'objet et l'outil ;
- un identifiant vers l'outil d'interaction qui a généré le message ;
- un identifiant unique.

Message Déplacer Lorsque l'outil d'interaction bouge dans ou sur un objet, un message "Déplacer" est envoyé. Celui-ci indique quel est le déplacement à la zone de contact entre deux événements de déplacement. Ainsi, lorsqu'un outil se déplace d'un bout à l'autre sur la surface d'un objet, une succession de messages "Déplacer" sont envoyés à l'objet. Ces messages correspondent à l'événement `onMouseMove` dans de nombreuses librairies $2D$.

Ainsi, chaque message "Déplacer" propose :

- le repère local correspondant à l'intersection entre l'objet et l'outil ;
- une information de déplacement depuis le dernier message déplacer ;
- un identifiant vers l'outil d'interaction qui a généré le message ;
- un identifiant unique.

Message Déposer Le message "Déposer" est envoyé par un outil d'interaction qui dépose un objet sur un autre objet. Par exemple, lorsqu'un rayon virtuel déplace une sphère virtuelle sur un cube, le cube reçoit, en plus du message "Entrer", un message "Déposer" qui lui indique qu'une sphère a été lâchée sur lui. Ce message permet de réaliser une action de glisser-déplacer, à la manière d'un environnement $2D$.

Ainsi, chaque message "Déposer" propose :

- le repère local correspondant à l'intersection entre l'objet et l'outil ;
- un identifiant vers l'outil d'interaction qui a généré le message ;
- un identifiant unique.

Signaux et slots

Définition Un couple signal - slot fournit un mécanisme simple et sécurisée de communication entre objets.

La plupart des interfaces graphiques nécessitent de pouvoir répondre aux interactions de l'utilisateur. Par exemple, lorsqu'il clique sur un bouton, l'application réalise une certaine action. Plus globalement, tout objet graphique d'une application doit pouvoir communiquer avec n'importe quel autre objet, quel que soit son type. De nombreuses librairies traitent ce problème en utilisant des mécanismes qui ne sont pas assez flexibles, pas sécurisés du point de vue du type d'objet[4], et souvent non orienté objet.

Dans notre exemple précédent, nous souhaitons associer l'exécution d'un code au clic de l'utilisateur. Sans signaux, il faut passer un pointeur de fonction au bouton, qui correspond à la fonction qui sera appelée lors de l'action. Avec cette méthode, il n'est pas possible de s'assurer que le type d'objet associé à ce pointeur est le bon ; c'est une source d'arrêt du programme qui l'utilise. L'autre problème avec cette méthode est qu'il existe un lien direct entre un composant graphique et la fonctionnalité associée, rendant difficile le développement de classes indépendantes.

Le mécanisme des signaux et des slots est un mécanisme de communication inter-objets qui peut remplacer complètement les anciens pointeurs de fonction et de messages utilisés précédemment dans de nombreuses librairies. Ils permettent de s'assurer du type des objets qui contiennent les signaux et les slots, et sont en outre flexibles et orientés objets. En reprenant notre exemple précédent, notre bouton "émet" un signal "cliqué" quand l'utilisateur clique dessus. Le programmeur peut ensuite choisir de connecter ce signal à une fonction (le "slot") qui sera associée. Les erreurs de type sont reportées au niveau de la compilation ; ceci évite la fin prématurée de l'application qui pourrait y être liée.

Dans notre exemple, nous souhaitons quitter l'application lorsque le bouton est enfoncé. Pour cela, l'application `app` contient une fonction `quit()`. Le bouton `bouton()` dispose du signal `clique()`. Il suffit d'écrire l'instruction suivant :

```
bouton->clique.connect(app, app->quit) ;
```

Des connexions et déconnexions supplémentaires peuvent être ajoutées ou enlevées pendant l'exécution de l'application. Il est également possible d'attacher plusieurs slots au même signal, et inversement plusieurs signaux au même slot. Par exemple, si l'application possède plusieurs boutons, et que le programmeur souhaite réaliser la même action quelle que soit le bouton enfoncé, il connectera chaque signal `clique()` à l'unique slot `quit()` :

```
bouton1->clique.connect(app, app->quit) ;
bouton2->clique.connect(app, app->quit) ;
```

[4] a. *type-safe*

La figure 4.3 montre l'exemple d'un objet déformé à l'aide d'une structure de déformation, intitulée `deform`. Lorsque l'utilisateur bouge un sommet, l'objet se déforme en conséquence, grâce à la connexion entre le signal `move()` du sommet `deform.Vi` et le slot `updateGeometry()` de l'objet `object`. De la même manière, pour changer le type du maillage de contrôle – par sommets, arêtes ou faces, il suffit de connecter le signal `click()` du groupe de boutons `ButtonGroup` au slot `changeType` du maillage de contrôle `deform`.

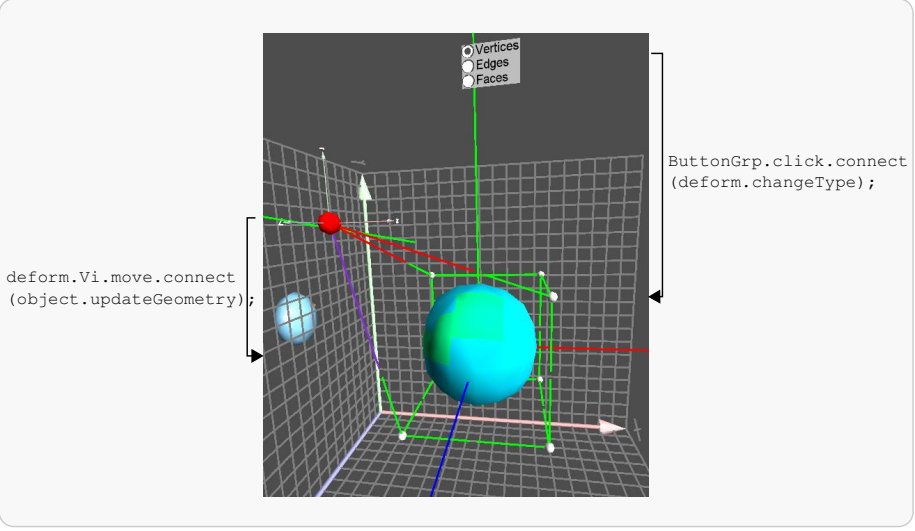


Fig. 4.3: Communication par signal/slot entre un sommet du maillage de contrôle et l'objet déformé (à gauche) et entre un groupe de boutons et le maillage de contrôle (à droite).

4.3.3 Graphe de scène

Les objets d'interaction – tels que les widgets – proposés par la vrLib sont contenus dans une structure de données arborescente de type graphe de scène acyclique orienté. Un tel graphe est une structure qui arrange la représentation spatiale et logique des objets entre eux au sein d'une scène graphique. L'intérêt d'une telle structure est de pouvoir contrôler, à n'importe quel niveau de la hiérarchie, les fils de ce niveau. Un graphe de scène ne comporte pas de cycle, ce qui lui évite de boucler indéfiniment. En réalité le graphe de scène est acyclique orienté. Lorsqu'une opération est appliquée à un nœud, ses effets sont automatiquement propagés à tous ses fils. Le principe n'est pas nouveau, et de nombreuses librairies comme OpenSG et OpenInventor proposent leur propre graphe de scène. Nous avons choisi de ré-écrire le nôtre afin d'éviter une dépendance inutile.

Chaque objet de la scène dispose d'un repère local qui est exprimé par rapport au parent. Il contient donc une matrice de transformation géométrique 4x4 en coordonnées homogènes, ce qui permet de l'afficher dans le repère de son propre père. Ainsi, lorsque l'utilisateur manipule et déplace un nœud qui contient un ou plusieurs fils, ces derniers sont automatiquement déplacés en

conséquence : à tout niveau, la matrice courante est concaténée à la matrice précédente. Par exemple, lorsque l'utilisateur déplace un widget Fenêtre, tous les widgets qu'elle contient – et qui sont ses fils, sont déplacés automatiquement de la même manière, comme s'ils étaient littéralement collés dessus. Le travail du développeur est facilité par cette gestion transparente de la hiérarchie des objets d'interaction.

Les avantages d'un graphe de scène au sein de la vrLib sont nombreux. Par rapport à l'affichage, une arborescence permet de gérer l'ordre d'affichage des objets de la scène et transmettre des informations d'un nœud père à ses fils (position, orientation, visibilité, ...). Par rapport à l'organisation de la scène, un tel graphe permet de regrouper de façon logique et intelligente les différents composants graphiques du programme. Chaque nœud est ainsi traité comme la racine de son propre sous-arbre. La structure du graphe de scène permet d'automatiser certains comportements qui viendront se propager dans la structure. Par exemple, lorsqu'un objet du graphe est supprimé, tous ses fils le sont également, ce qui facilite la gestion de la mémoire pour le programmeur.

Cette section est centrée sur la gestion des objets graphiques au sein du graphe de scène de la librairie vrLib. Nous présentons d'une part la gestion des objets de la librairie : quels sont les rapports qu'ils entretiennent avec le matériel ? quels gestionnaires sont mis en œuvre pour contraindre ces objets et, éventuellement, leur associer un effet graphique, et quels mécanismes sont utilisés pour séparer aspect graphique et comportement ? D'autre part, nous présentons les principaux widgets que propose la librairie : les conteneurs, les boutons, ceux dédiés à l'entrée de données, et enfin ceux consacrés à l'affichage de données.

Comportement et graphisme des objets

Tout objet du graphe de scène de la vrLib dispose par défaut d'un comportement simpliste qui ne lui permet que d'être positionné et déplacé librement dans l'environnement virtuel. La librairie propose deux mécanismes qui sont dédiés à la définition et à la gestion de contraintes de manipulation *via* le gestionnaire de contraintes, et à la mise en œuvre d'effets graphiques *via* le gestionnaire d'effets graphiques. De plus, nous proposons une méthode de séparation entre la programmation du comportement d'un objet, et la création de son apparence graphique.

Gestionnaire de contraintes Un objet de la vrLib dispose d'un comportement de base : il peut être déplacé librement dans la scène virtuelle. Cependant, il existe des situations dans lesquelles il est nécessaire de contraindre les mouvements possibles. Par exemple, imaginons qu'une scène virtuelle dans laquelle une sphère roule le long d'un plan incliné. Cet objet sphérique pourrait contenir le code nécessaire au comportement que le programmeur souhaite lui conférer. Cependant, pour tout autre objet au comportement identique, il faut dupliquer les contraintes de déplacement.

La vrLib propose de déporter les contraintes hors des objets, dans un gestionnaire. Celui-ci aura pour rôle d'associer pour tout objet souhaité, une contrainte

particulière. La gestion de cette contrainte peut éventuellement être confiée à une librairie externe comme un solveur de contraintes géométriques, ou encore une librairie physique – gravité, poids, etc. Ainsi, le gestionnaire de contraintes indique exactement à tous les objets contraints comment il doivent se déplacer dans la scène virtuelle.

Le principe général du gestionnaire de contraintes est la projection de l'outil d'interaction dans le lieu géométrique défini par un objet. Par exemple, avec l'outil du lancer de rayon, il s'agit de l'intersection entre le rayon et le lieu géométrique d'un objet. Par défaut, la vrLib propose plusieurs contraintes simples : le déplacement le long d'un axe, sur un plan ou une sphère, la rotation autour d'un axe, d'un point ou d'un cylindre.

En pratique, le gestionnaire de contraintes repose sur un motif de conception de la factorisation. L'intérêt de cette méthode est de déléguer au gestionnaire des contraintes détachées des objets sur lesquels elles doivent s'appliquer. Ceci permet d'obtenir une architecture évolutive, réutilisable, quelle que soit la manière de résoudre les contraintes – solveur géométrique, moteur d'effets physiques, etc.

Gestionnaire d'effets graphiques La vrLib introduit la notion d'effets graphiques sur les objets $3D$ du graphe de scène. Par exemple, nous souhaitons que lorsque l'utilisateur passe son outil d'interaction sur une icône $3D$, cette dernière se mette à flotter et à tourner dans la scène virtuelle, jusqu'à ce que l'outil quitte l'objet. Tous les effets graphiques appliqués aux objets de notre librairie sont pris en charge par le gestionnaire d'effets graphiques.

Le principe de ce gestionnaire est très similaire au gestionnaire de contraintes vu plus haut. La principale différence est qu'un effet graphique ne modifie pas la position réelle d'un objet et sa durée est paramétrée par le programmeur.

Comme pour le gestionnaire de contraintes, il est possible de déléguer un ou plusieurs effets graphiques à une librairie externe grâce à la méthode de factorisation mise en œuvre. La vrLib propose plusieurs effets différents : rebond sur place, rotation simple centrée sur l'objet, et la dilatation suivie de la contraction.

Du code à la représentation graphique Un de nos objectifs est de pouvoir séparer le code de la représentation graphique d'un objet. Les objets déjà inclus dans la librairie proposent un comportement par défaut et un aspect graphique prédéfini. Notre volonté est de séparer autant que possible le travail du programmeur et le travail du graphiste, comme cela existe pour les interfaces $2D$, afin de pouvoir changer l'apparence des objets de la librairie sans avoir à modifier le code du comportement.

Le programmeur a en charge la conception logicielle des composants graphiques et de l'interaction ; son rôle ne doit en aucun cas être celui d'un graphiste. Inversement, le créateur de l'interface n'a, *a priori*, aucune connaissance de la programmation, et son rôle doit se limiter à concevoir l'aspect visuel des objets. La séparation entre comportement et graphisme constitue une problématique très intéressante. Par exemple, tous les menus servent à explorer un

arbre d'options en affichant un certain nombre d'éléments ou simplement un sous-menu. Cependant, leur aspect peut varier énormément – menu $2D$, C^3, Ring. Le programmeur écrira un minimum de code permettant uniquement de gérer son comportement, quel que soit le menu. Et le graphiste aura la charge de dessiner la représentation visuelle de ce menu.

Nous proposons une solution partielle à ce problème : plutôt que de décrire l'apparence des objets graphiques dans le code de la vrLib, et obliger le concepteur à programmer un aspect prédéfini à l'aide de commandes OpenGL, il doit pouvoir utiliser directement les objets $3D$ réalisés dans un modeleur par un graphiste.

La première étape de notre méthode est la définition par le graphiste d'un maillage A, à l'aide d'un modeleur géométrique. Ce maillage donne la géométrie exacte du futur objet vrLib que le programmeur va coder. Normalement, dans le modeleur, en sélectionnant un ensemble de triangles, le graphiste peut définir un sous-maillage nommé. Nous détournons ce mécanisme pour associer un nom à ce sous-maillage – par exemple `ZONE1`. Ce nom est maintenant utilisé dans 2 cas.

Le premier rôle de l'ensemble de triangles, est d'identifier une zone, au sens de la vrLib, et d'y associer un certain comportement. Par exemple, le graphiste dessine une fenêtre dans son modeleur, puis il sélectionne un ensemble de triangles qu'il nomme `QUITTER`. Ensuite, le programmeur charge ce maillage fenêtre dans un objet vrLib nommé par exemple `Fenetre`. Puis, il programme un comportement particulier sur une zone nommée `QUITTER`, pour fermer la fenêtre. Ce mécanisme nous permet ainsi d'avoir un comportement unique, quel que soit le maillage dessiné, du moment que le graphiste a associer le bon nom de zone à un ensemble de triangles.

Le second rôle de l'ensemble de triangles est d'arriver à définir un repère local à la surface du maillage dessiné par le graphiste, pour l'objet vrLib A. Nous nommons ce repère une ancre. Ainsi, il est possible de placer automatiquement et correctement un autre objet vrLib B sur le maillage de A. Par exemple, le graphiste dessine une fenêtre sphérique, puis sélectionne un ensemble de triangles qu'il nomme `ANCRE1`. Le programmeur peut maintenant coder un objet vrLib `Fenetre` et un objet `Bouton`. En associant l'objet `Bouton` à l'ancre `ANCRE1` de l'objet `Fenetre`, le bouton se placera correctement à la surface de la fenêtre. La figure 4.4 illustre le placement d'un bouton sur la surface d'un maillage dessiné dans un modeleur géométrique, *via* une ancre.

Ce second mécanisme permet de définir des ancres sur un maillage, c'est-à-dire un repère local sur la surface d'un objet de la vrLib. Dans notre exemple précédent, la fenêtre étant sphérique, le bouton est visible, même s'il n'épouse pas complétement la géométrie de la fenêtre. En revanche, avec un maillage concave, le bouton pourrait disparaître à l'intérieur du maillage de la fenêtre. La vrLib n'adapte pas le maillage d'un objet à la géométrie d'un autre objet ; c'est au graphiste de s'assurer que les objets qu'ils modélisent donneront des résultats visuels satisfaisants.

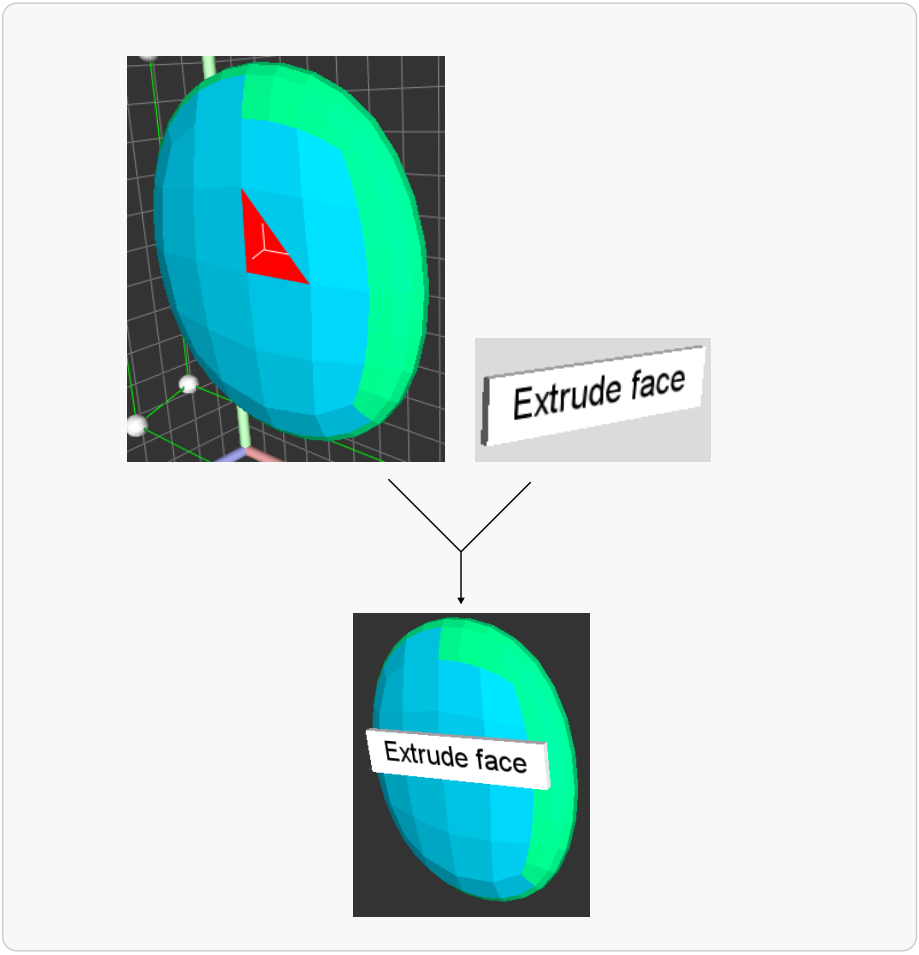


Fig. 4.4: Accroche d'un bouton à la surface d'une maillage *via* une ancre définissant un repère local.

Widgets

Dans cette section, nous allons décrire quelques widgets importants offerts par vrLib. La figure 4.5 illustre la hiérarchie globale des widgets qui sont utilisés par les programmes reposant sur cette librairie, et qui sont inspirés de ceux rencontrés en environnement $2D$. Nous pouvons voir que la classe dont héritent tous les widgets s'intitule `Component`. Cette classe contient toutes les informations nécessaires à un objet de la vrLib, comme par exemple des attributs de visibilité, de positionnement et d'orientation dans l'espace, ainsi que la gestion d'enregistrement dans la hiérarchie globale de la librairie. Chaque `Component` comporte à la fois un identifiant unique qui permet de le retrouver dans la hiérarchie, et un nom pour l'appeler explicitement. L'orientation et le positionnement d'un `Component` peut s'exprimer soit globalement par rapport au repère de la scène, soit localement par rapport à un autre `Component` et dans ce cas, lorsque l'objet parent est déplacé, ses fils sont déplacés de la même manière.

Un objet `Widget` dérive des classes `Animable` et `Draggable`, qui lui confèrent un comportement lors de la manipulation, et éventuellement un effet graphique. Nous avons vu plus haut comment il est possible de manipuler et animer à loisir un objet de type `Widget`. Nous notons que tous les widgets, s'ils sont re-dimensionnables, intègrent un mécanisme qui permet de cacher automatiquement toutes les informations qui dépassent visuellement.

Nous pouvons répartir les widgets qui sont présents dans la vrLib en quatre groupes. Les conteneurs constituent le premier groupe, puisque leur rôle est dévolu à contenir et regrouper d'autres objets `Widget`. Le second groupe est composé des boutons qui permettent de lancer la ou les commandes qui y sont associées. Les widgets du troisième groupe servent à rentrer des données, qu'il s'agisse de nombres, de lettres ou encore d'un dessin réalisé à main levée. Enfin, le quatrième et dernier groupe propose des widgets dédiés à l'affichage des données. Regardons plus attentivement chacun de ces widgets.

Conteneurs Les widgets conteneurs servent à regrouper d'autres widgets ; on trouve à la fois des composants graphiques issus du monde $2D$, comme les fenêtres, les groupes de boutons, les menus plats, et aussi des composants plus spécifiques à un environnement $3D$, comme les barres de rangement $3D$ et les calques $3D$.

Fenêtre (`MainWindow`) Le conteneur fenêtre ou `MainWindow` contient un ensemble de widgets standards, comme les boutons, les menus, etc. Le haut de la fenêtre est occupé par une barre de titre comprenant une icône, un titre, et trois boutons pour miniaturiser, agrandir et fermer la fenêtre. L'espace situé en-dessous de cette barre de titre comprend une barre de menu.

Il est également possible de modifier dynamiquement la taille de la fenêtre en accrochant le bord gauche, bas ou droit, de la même manière que dans un environnement $2D$. Pour déplacer ce widget, il suffit de sélectionner la barre de titre, par exemple avec le lancer de rayon. L'initialisation peut se faire dans le code du programme directement ou être décrit directement dans un fichier de description d'interface. La figure 4.6 présente ce widget.

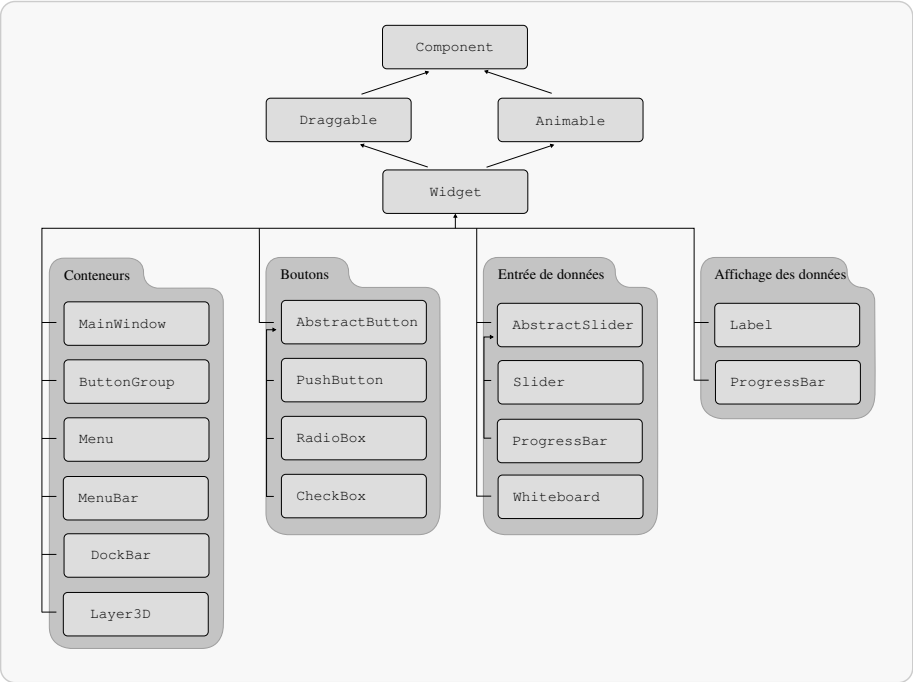


Fig. 4.5: Hiérarchie globale des widgets de la vrLib.

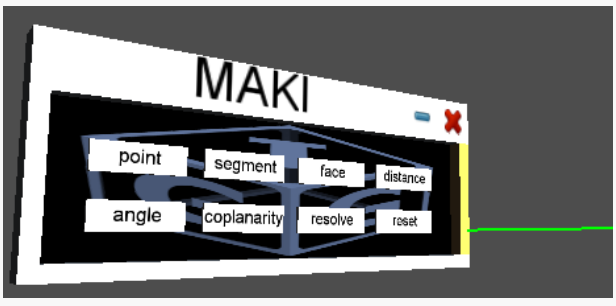


Fig. 4.6: Widget `MainWindow` de la vrLib.

Groupe de boutons (`ButtonGroup`) Le widget `ButtonGroup` permet de regrouper et d'organiser un ensemble de boutons. La particularité de ce conteneur est de rassembler, au sein d'une même et seule entité, plusieurs boutons dont la fonctionnalité est similaire.

Un groupe de boutons peut être exclusif ; dans ce cas, un seul bouton peut être actif à la fois. Chaque bouton contenu dans un groupe de ce type possède un identifiant unique, spécifique à cet ensemble de boutons. Lorsque l'utilisateur décide de cliquer sur l'un des boutons du groupe, un message `clicked()` est émis avec l'identifiant unique du bouton correpondant. La figure 4.7 montre le conteneur `ButtonGroup`.

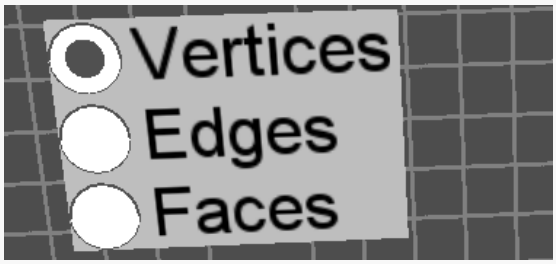


Fig. 4.7: Widget `ButtonGroup` de la vrLib.

Menu plat (`Menu`) Le widget `Menu` est un composant conteneur dont le rôle est d'organiser une arborescence d'éléments, que l'on nomme des `MenuItem`. Un menu peut également contenir d'autres menus de même nature, et l'on parle alors de menu hiérarchique. À chaque instant de son utilisation, seule la partie de l'arborescence que l'utilisateur explore est affichée, ce qui évite d'encombrer inutilement l'espace visuel de travail. Un menu est généralement rattaché à une fenêtre (`MainWindow`), et peut, plus rarement, flotter librement dans l'espace. La figure 4.8 présente le widget `Menu`, affiché depuis une barre de menus.

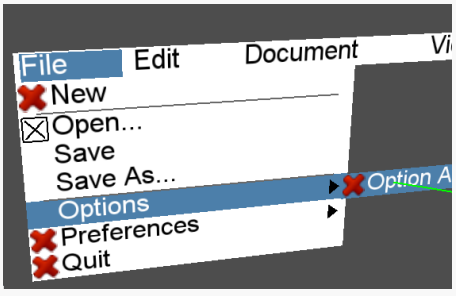


Fig. 4.8: Widget `Menu` de la vrLib.

Barre de menus (`MenuBar`) Dans une interface graphique, la barre de menus est une bande horizontale située dans la partie supérieure d'une fenêtre qui contient les titres des menus disponibles. En cliquant sur un titre de menu, l'utilisateur peut sélectionner une entrée dans le menu qui vient de se dérouler et d'apparaître. La figure 4.8 montre un menu plat surmonté d'une barre de menus.

Barre d'icônes $3D$ (`DockBar`) Une barre d'icônes $3D$ permet de miniaturiser un widget de l'interface graphique en une représentation symbolique $3D$, à la manière de MacOS X d'Apple. Ainsi, le widget miniaturisé n'occupe qu'une place limitée, ce qui évite d'encombrer inutilement l'espace de travail, tout en restant facilement accessible. Le widget `DockBar` fonctionne donc comme un menu de widget, et chaque clic sur un élément permet de restaurer l'état initial du widget correspondant.

Ce widget est un anneau hémisphérique dont une seule partie est visible à un instant donné. Deux contrôles, représentés graphiquement par des flèches permettent de faire défiler les éléments d'un côté ou de l'autre de la barre. La figure 4.9 illustre le widget `DockBar`.

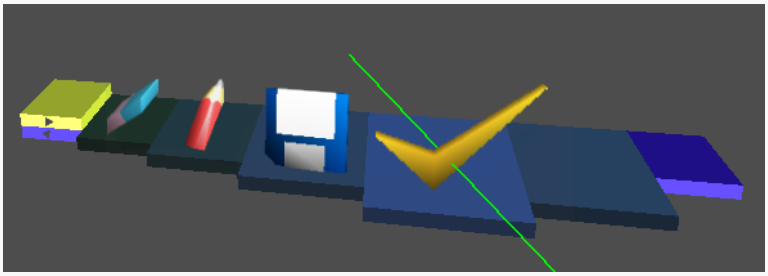


Fig. 4.9: Widget `DockBar` de la vrLib.

Calque $3D$ (`Layer3D`) Un calque $3D$ permet de regrouper un ou plusieurs widgets, de même type ou non, au sein d'un seul conteneur. À la manière des calques d'un dessinateur, ce widget offre deux fonctionnalités intéressantes. D'une part, un calque peut être actif, dans ce cas tous les éléments qu'il contient sont manipulables et affichés, ou inactif, et dans ce cas aucun élément n'est affiché et manipulable. D'autre part, il est possible de manipuler tous les éléments d'un coup, en travaillant localement sur le calque : le calque agit ainsi comme s'il était le père de tous les widgets qu'il regroupe. La figure 4.10 montre le widget `Layer3D`.

Boutons Les boutons sont des composants graphiques que l'on actionne en cliquant dessus, *via* un outil d'interaction tel que le lancer de rayon. La vrLib fournit trois types de boutons différents : les boutons pressoirs, les boutons radio, et les cases à cocher. Notons que plusieurs boutons peuvent être regroupés au sein d'un groupe de boutons. Il s'agit du composant graphique de base de toute

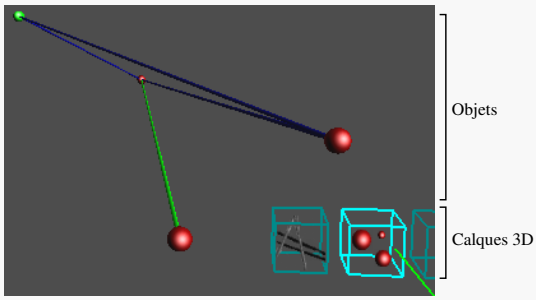


Fig. 4.10: Widget `Layer3D` de la vrLib.

interface graphique, grâce à laquelle l'utilisateur peut lancer des actions. La figure 4.11 donne un aperçu de ces trois boutons disponibles dans la vrLib.

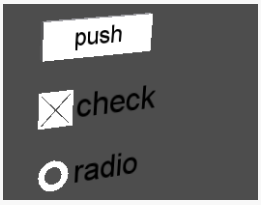


Fig. 4.11: Widgets `PushButton`, `RadioButton` et `CheckBox` de la vrLib.

Bouton poussoir (`PushButton`) Le widget `PushButton` est le bouton le plus simple. En cliquant dessus à l'aide d'un outil d'interaction, il passe de l'état relâché à l'état enfoncé, et la ou les actions correspondantes sont activées. Un texte ou une image, affiché à la surface du composant, permet d'associer une description concise de la fonction du bouton. Par exemple, en appuyant sur un bouton avec le texte `allumer`, une lumière sera activée dans la scène virtuelle. Par défaut, un widget `PushButton` ne possède que deux états, enfoncé ou relâché, mais il est possible de lui attribuer un troisième état intermédiaire.

Bouton radio (`RadioButton`) Le widget `RadioButton` est un bouton exclusif ; en effet, lorsque plusieurs boutons radio sont groupés, un seul peut être enfoncé à un instant donné, tant qu'un autre n'aura pas été cliqué. Ce type de widget permet de faire un seul et unique choix parmi N possibles.

Cases à cocher (`CheckBox`) Les cases à cocher sont des boutons à état du même type qu'un `RadioButton`, et peuvent être également exclusifs lorsqu'il sont regroupés à d'autres widgets `CheckBox`. À la différence d'un bouton radio qui permet faire 1 choix parmi N, une case à cocher sert essentiellement à faire N choix parmi M.

Entrée de données Les composants d'entrée de données ont pour fonction de manipuler des valeurs, ou plus globalement des symboles (caractères, chiffres, ...). La vrLib propose trois widgets dédiés à entrer des données : la ligne de texte, la barre de défilement, et la zone de dessin $2D$.

Ligne de texte (`TextLine`) Un composant `TextLine` sert à l'utilisateur à entrer et éditer une simple ligne de texte à l'aide d'un ensemble de fonctions, comme par exemple les opérations "annuler" et "refaire", et "copier" et "coller". Il s'agit d'un widget que l'on utilise en combinaison avec un clavier pour manipuler des chaînes de caractères. Lorsqu'aucun clavier physique n'est disponible, par exemple lorsque l'environnement n'en offre pas, on pourra utiliser un outil d'interaction dédié à l'entrée de symboles.

Barre de défilement (`Slider`) Une barre de défilement est un widget affiché verticalement, horizontalement ou de biais, qui permet de contrôler une valeur bornée. Elle permet à l'utilisateur de glisser un curseur le long d'une barre qui représente l'intervalle. Le `Slider` se manipule directement avec le curseur. Par défaut, une barre de défilement peut être orientée de trois manières différentes : horizontalement comme le montre la figure 4.12, verticalement ou encore de biais.

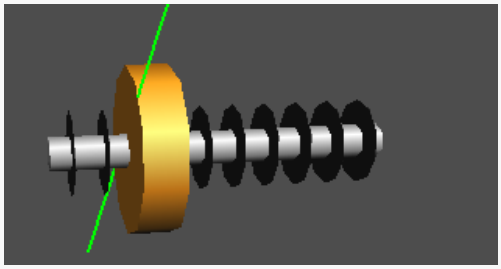


Fig. 4.12: Widget `Slider` de la vrLib.

Zone de dessin $2D$ (`BlackBoard`) Une zone de dessin est un plan sur lequel l'utilisateur peut dessiner à l'aide de l'outil d'interaction. Par exemple, l'outil du lancer de rayon peut servir à laisser une trace sur le `BlackBoard`, à la manière d'un dessinateur sur une feuille de papier. Par défaut, le croquis est enregistré dans une texture qui est maintenue affichée à la surface de composant. Une extension de ce widget permet d'empiler plusieurs calques $2D$, de manière à pouvoir supersposer les dessins. Nous pouvons voir un exemple d'utilisation du composant `BlackBoard` mono-calque sur la figure 4.13.

Affichage de données Certains composants graphiques sont consacrés à la présentation de données de l'application. Ils fournissent peu de possibilités d'interaction. La vrLib propose deux widgets : l'étiquette et la barre de progression.

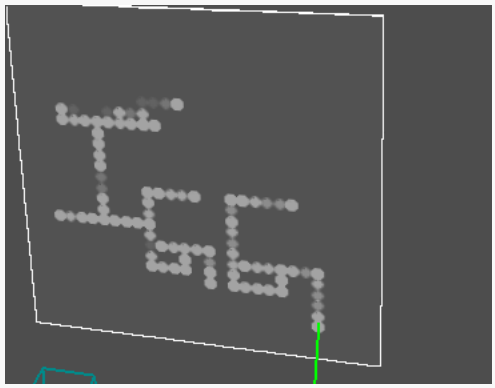


Fig. 4.13: Widget `BlackBoard` de la vrLib.

Etiquette (`Label`) Le widget étiquette permet d'afficher du texte ou une image. Aucune fonctionnalité d'interaction n'est disponible pour l'utilisateur, puisque sur rôle est dévolu à l'affichage d'informations. La figure 4.14 montre une simple étiquette avec le message "Hello world !".

Fig. 4.14: Widget `Label` de la vrLib.

Barre de progression (`ProgressBar`) Le widget `ProgressBar` est employé pour donner à l'utilisateur une indication de la progression d'une opération et lui indiquer que l'application est encore en cours d'exécution, comme par exemple l'état d'un calcul intensif.

La barre de progression repose sur le concept d'étapes ; le programmeur donne le nombre total d'étapes et le nombre d'étapes déjà accomplies, et le composant affiche le pourcentage courant réalisé. La figure 4.15 illustre le widget `ProgressBar`.

4.3.4 Outils d'interaction

Les outils d'interaction permettent d'agir sur les objets d'une scène virtuelle. Sélectionner, manipuler, naviguer, interagir avec l'interface ou simplement entrer des symboles, chaque technique permet de contrôler l'application – plus globalement le système – et les données qui sont liées. La vrLib introduit la notion d'outil d'interaction. Nous expliquons dans une premier temps le fonctionnement des outils d'interaction. Puis, dans un deuxième temps, nous introduisons

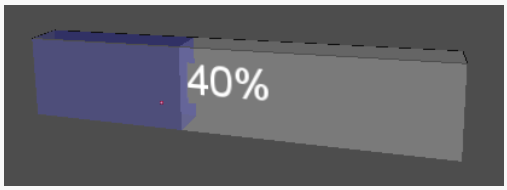


Fig. 4.15: Widget `ProgressBar` de la vrLib.

la notion d'interaction adaptative, qui constitue une des particularités de notre librairie.

Fonctionnement des outils

Les outils d'interaction de la vrLib utilisent par défaut le mécanisme OpenGL de *picking*. Son principe de fonctionnement est assez simple : chaque objet est dessiné à l'écran afin de constituer visuellement une partie de la scène virtuelle. Puis, il est dessiné une seconde fois, mais sans produire de pixels. L'objet ne s'affiche pas à l'écran, au lieu de ça, les informations de dessin sont retournées à l'application. En mode sélection, l'objet définit un ou plusieurs noms qui encadrent des instructions de dessin. Tout outil d'interaction définit un volume de sélection en forme de pyramide tronquée. Lorsqu'un objet qui porte un ou plusieurs noms tombe dans le volume de sélection, la liste des noms qu'il contient est renvoyée à la vrLib. Celle-ci effectue alors plusieurs opérations pour aboutir à un comportement spécifique. La figure 4.16 illustre le fonctionnement du *picking* OpenGL : seuls les deux plus gros objets sphères sont sélectionnés par le rayon virtuel.

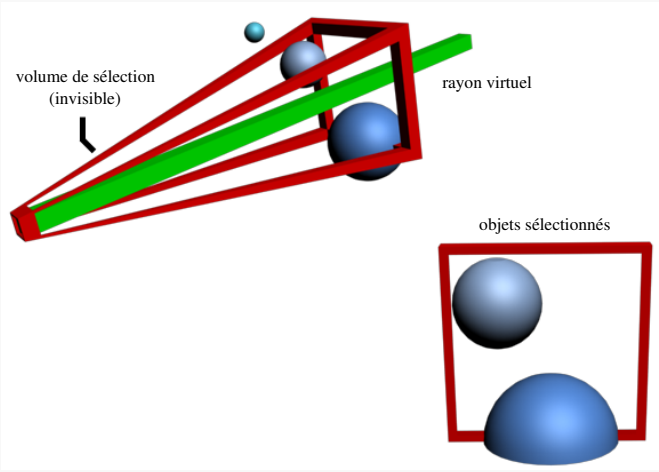


Fig. 4.16: Mécanisme de sélection OpenGL utilisé par les outils d'interaction.

L'avantage de ce mécanisme est qu'il est simple et rapide, étant géré directement par le matériel graphique, mais l'inconvénient majeur est qu'il est très difficile, voire impossible, de créer des volumes de sélection "exotiques", tels qu'une sphère. Dans ce cas, la vrLib a été conçue pour abstraire le mécanisme de sélection utilisé, et il est tout à fait envisageable de recourir à un *picking* logiciel.

À tout outil d'interaction est associé un ou plusieurs traqueurs, comme par exemple le traqueur posé sur la main de l'utilisateur. Ceux-ci servent à déplacer le volume de sélection dans l'espace. Tout outil peut contenir de la même manière plusieurs événements de déclencheur et de valuateur *via* une couche d'abstraction matérielle, comme par exemple l'enfoncement d'un bouton ou encore un geste déclencheur réalisé par l'utilisateur.

Définition d'un comportement sur les objets Afin de déterminer quel comportement on souhaite associer aux objets en fonction des comportements de l'utilisateur, chaque outil contient 3 listes d'objets. Ces listes sont utilisées pour envoyer, à chaque itération de l'application, les messages systèmes adéquats. Il s'agit des :

- éléments vus par le *picking* (désignés) ;
- éléments vus par le *picking* à l'itération précédente (précédemment désignés) ;
- éléments capturés, qui continuent à être désignés, mais qui ne reçoivent plus les messages "Entrer" et "Sortir".

Nous utilisons le terme d'élément plutôt que d'objet, car tout objet peut définir éventuellement une ou plusieurs zones. Par exemple, un objet fenêtre graphique peut être défini avec un élément – aussi nommé zone – bouton "Fermer". Il y aura donc un comportement sur la fenêtre, et sur le bouton, bien qu'il s'agisse d'un seul et même objet.

Ces trois premières listes permettent d'en déterminer trois autres, plus évoluées :

- éléments dans lesquels l'outil d'interaction vient d'entrer (entrer = désignés - précédemment désignés) ;
- éléments dans lesquels l'outil d'interaction vient de sortir (sortir = précédemment désignés - désignés) ;
- éléments dans lesquels l'outil d'interaction se trouve encore par rapport à l'itération précédente (encore désignés = désignés - entrer).

Le mécanisme de *picking*, en plus de retourner une liste des noms d'éléments graphiques qui tombe dans le volume de sélection, retourne également une information de distance entre le traqueur et l'élément désigné. L'outil d'interaction calcule alors une matrice d'intersection entre lui-même et l'élément désigné. Cette information est ensuite ajoutée à tous les messages que l'outil d'interaction renvoie aux objets qu'il manipule. Cette information de distance permet donc de savoir à quel endroit se trouve l'outil sur l'élément désigné au

moment de l'envoi du message.

Les 6 listes que nous venons de voir, combinées aux autres événements de l'outil – déclencheur, valuateur, etc. – permettent de déterminer l'ensemble des messages système que l'outil envoie aux éléments qu'il désigne. Ainsi, chaque outil permet d'interagir avec les objets d'une application. Les objets proposés par la vrLib, comme par exemple les widgets, disposent d'un comportement par défaut, ce qui évite au programmeur la tâche de définition d'un comportement. La souplesse de notre librairie permet évidemment de redéfinir chaque comportement, soit au niveau de l'objet, soit au niveau de l'application. En outre, il est possible de définir le même comportement pour tous les composants graphiques d'un même type préalablement enregistré. Par exemple, nous disposons de plusieurs objets qui héritent les uns des autres, en partageant certaines données et fonctionnalités. En utilisant ce mécanisme sophistiqué de définition d'un comportement par type d'objet, il est possible de n'écrire le comportement qu'une seule fois.

Précisons qu'il est possible de garder le contrôle sur un objet sélectionné, même si l'outil d'interaction ne le touche plus : il s'agit du mécanisme de capture. Par exemple, lorsqu'une application ralentit pour cause de calculs importants, l'outil peut sortir d'un objet en cours de manipulation. Prenons également un autre exemple en imaginant que l'utilisateur attrape une fenêtre pour la bouger, et que lorsqu'il l'attrape elle se mette à tourner sur elle-même *via* un effet graphique. Il peut arriver un moment où l'outil d'interaction ne touche plus cette fenêtre. Sans le mécanisme de capture, les messages "Entrer" et "Sortir" seraient générés, alors qu'ils ne devraient pas l'être.

Bulles d'information Le *Tool Tip* (ou *Tool Tip Text*) est une bulle d'information qui s'affiche lorsque l'utilisateur passe l'outil sur un objet. En effet, chaque objet peut définir un texte qui sera affiché dans cette bulle. Elle s'affichera pendant quelques secondes en face de l'utilisateur, après que ce dernier ait placé son outil d'interaction sur l'objet. Puis, elle disparaîtra sans obstruer inutilement le reste de la scène virtuelle. Cette fonctionnalité est très utile pour aider l'utilisateur dans son interaction. Il est également possible de définir une aide sonore, mais nous ne mettons pas en œuvre ce mécanisme dans nos programmes parce qu'il devient rapidement gênant ; en effet, la successions des informations sonores peuvent déconcentrer l'utilisateur.

Interaction adaptative

La notion d'interaction adaptative découle directement de notre réflexion sur l'interaction conduite par la dimension ; celle-ci est décrite dans le chapitre 2 de cette partie. Nous décrivons ici la philosophie et la mise en œuvre au sein de la vrLib d'un tel mécanisme permettant à l'application de changer automatiquement d'outil en fonction de l'objet sur lequel l'utilisateur travaille.

Philosophie Nous souhaitons disposer d'outils d'interaction qui changent selon l'objet sur lequel ils se trouvent. Par exemple, l'utilisateur manipule l'outil d'interaction Laser sur certains objets de la scène, mais lorsqu'il fera un choix sur un menu $2D$, ce paradigme se transformera en un curseur $2D$ contraint dans

le plan du menu.

Soient un maillage et deux utilisateurs, u_1 et u_2, qui travaillent indépendamment dessus, et sur deux endroits différents. Ce maillage est l'objet, et l'utilisateur u_1 déplace les sommets du maillage librement dans l'espace – 6 degrés de liberté, alors que l'utilisateur u_2 déplace les sommets selon l'axe u_x de la scène virtuelle uniquement. D'un côté, u_1 utilise l'outil d'interaction m_1, et de l'autre l'utilisateur u_2 manipule le maillage à l'aide de l'outil m_2. Il faut donc qu'on puisse également associer plusieurs paradigmes d'interaction différents pour chaque objet.

Par défaut, toute application définit un outil d'interaction initial qui permet d'interagir avec l'ensemble des objets de la scène virtuelle. Il s'agit de l'outil disposant du plus grand nombre de degrés de liberté possible : le lancer de rayon. Il agira sur tout objet qui ne définit pas un autre outil. Dès lors qu'un autre paradigme est associé, l'application utilisera celui-là à la place du rayon par défaut, tant que le nouvel outil se trouve sur l'objet.

Mise en œuvre Etudions maintenant de plus près l'architecture interne à la vrLib, qui permet de changer d'outils d'interaction à la volée. L'application maintient à jour :

- la liste de l'ensemble des outils d'interaction disponibles ;
- la liste de l'ensemble des outils d'interaction utilisés à un moment donné du déroulement de l'application ;
- une table d'association, qui, pour un objet, retourne une liste d'outils d'interaction à utiliser.

Pour chaque outil d'interaction, l'algorithme d'envoi des messages est identique :

- *picking* ;
- envoi des messages de chaque outil à tous les objets concernés.

Nous ajoutons une étape intermédiaire à cet algorithme afin de prendre en compte la gestion des nouveaux outils d'interaction. L'algorithme devient alors :

- *picking* ;
- mise à jour des outils d'interaction :
  1. l'outil courant se remplace lui-même, pour chaque objet, par le nouvel outil, s'il existe ;
  2. l'outil courant copie ses attributs dans le nouvel outil ;
  3. l'outil courant est supprimé de la liste des outils utilisés ;
  4. l'outil remplaçant est ajouté à la liste des outils utilisés ;
  5. le nouvel outil génère ses propres messages à partir des informations de l'outil qu'il remplace ;
- envoi des messages de chaque outil à tous les objets concernés.

Nous décrirons un peu plus loin, en section 4.4, l'écriture par le programmeur d'une application utilisant ce mécanisme ; pour un objet de l'application, l'outil par défaut est remplacé par un nouvel outil. La vrLib fournit en standard les outils lancer de rayon, ainsi que le curseur $2D$ que l'on distingue sur la figure 4.19. D'autres techniques d'interaction seront traitées dans le chapitre 2 de cette partie.

4.3.5 Environnement de développement

La vrLib a été écrite entièrement en C++ ANSI, et ne dépend que d'un nombre limité de librairies : OpenGL et GLU pour la partie graphique, Slotsig pour les signaux mais qui est inclue en interne, et Xerces-c pour les fonctionnalités XML. Nous souhaitons remplacer, à court terme, Xerces-c au profit d'un parseur interne, afin de limiter les dépendances et d'alléger et de simplifier l'utilisation d'XML pour le programmeur.

L'ensemble de la vrLib représente environ 70 000 lignes de code, hors applications. Construire une application de Réalité Virtuelle demeure pour le moment assez complexe. Il n'existe pour le moment aucun standard matériel, et aucun standard logiciel sur lesquels s'appuyer pour développer un tel logiciel. Nous avons donc choisi de définir et nous conformer à nos propres guides de développement. Il fallait en particulier autoriser la modularité, la flexibilité et l'évolutivité de la librairie. La mise en œuvre des motifs de conception permet de résoudre de nombreux problèmes, comme par exemple l'interfaçage avec d'autres librairies, telles VRJuggler ou CAVElib. Nos motifs de conception les plus utilisés ont été la Fabrique, le Singleton, l'Objet Composite et la Stratégie. Une librairie ne peut exister sans une documentation adaptée, et vrLib ne déroge pas à la règle : nous proposons près de 200 pages de documentation détaillée.

Prise en charge du matériel et de la stéréovision

Notre librairie a été testée essentiellement sous Linux, mais elle compile et tourne également sous Windows, sachant que nos programmes de test ont été écrits pour l'API VRJuggler ; cette dernière étant bien plus restrictive que la vrLib, puisque le nombre de dépendances externes est plus important. Une librairie comme VRJuggler est nécessaire puisque la vrLib ne gère pas la configuration matérielle de l'affichage stéréoscopique. Notre environnement de travail et de développement se compose d'un *Workbench* BARCO de 2 écrans et du système de suivi de mouvements par caméras infrarouges ARTtrack2 d'ART. L'ensemble repose sur un cluster de 30 bi-processeurs Itanium 2 à 1.3 GHz tournant sous Linux. Les machines sont interconnectées par un réseau Myrinet 2000 (2 Gbits/s).

Tout utilisateur doit porter une paire de lunettes à obturation rapide qui permet au cerveau de reconstituer une image stéréoscopique à partir de deux images légèrement décalées. Nous avons également à disposition 3 types de gants de données (les P5 à 5 angles de flexion d'Essential Reality, le X-ist à 15 angles de flexion de NoDNA et le 5DT à 5 angles de flexion de Fifth Dimension), 1 joystick $3D$ Flystick de la société ART. La figure 4.17 donne la liste du matériel

que nous utilisons.

Fig. 4.17: Matériel utilisé pour implanter la vrLib.

La librairie VRJuggler nous permet dans une premier temps de coder et tester une application sur station de travail standard, *via* le mode de simulation. L'application est ensuite compilée sur le *Workbench* afin de réaliser des tests d'interaction grandeur nature et de terminer la mise au point de l'application. Les gants de données P5 ont été utilisés sur station de travail, alors que nous avons retenu les X-ist sur le *Workbench*. Ces derniers, avec 15 angles de flexion et 5 capteurs de pression au bout des doigts sont les gants les plus précis. Cependant, comme ils ne possèdent pas de système de suivi intégré, contrairement aux P5, il est impossible de les utiliser sur station, sauf à tester certaines postures.

Gestion d'un environnement multi-contexte

Notre environnement de développement repose sur un ensemble de machines et de nœuds graphiques reliés entre eux *via* une connexion réseau. VRJuggler prend en charge le calcul de la stéréovision sur l'ensemble du cluster que nous utilisons, à partir d'une scène OpenGL traditionnelle – ce code ne contient aucune instruction spécifique à la stéréo ou au matériel. OpenGL est une librairie qui fonctionne sur une machine à états, et chaque fenêtre de rendu qui utilise OpenGL a une machine à états qui est associée. Une telle association est aussi appellée un contexte de rendu OpenGL.

Chaque contexte maintient les état internes d'une instance OpenGL. Parmi ces états se trouvent :

- la couleur courante ;
- le mode d'éclairage courant ;
- la texture courante ;
- les listes d'affichage ;
- les objets texture.

Dans une application à un seul contexte de rendu, lorsque le programmeur souhaite ajouter une texture à un objet, il doit d'abord générer un identifiant

unique de texture à l'aide d'une commande OpenGL, qui sera stockée et utilisée par la machine à état propre à ce contexte. Ce mécanisme simple pose cependant des problèmes lorsque l'on souhaite passer à un environnement multi-contextes, tel que le nôtre sur le *Workbench*.

Nous souhaitons que l'utilisateur puisse programmer son code sans avoir à se soucier de l'architecture matérielle sous-jacente. Or, lorsqu'il y a plusieurs contextes graphiques qui sont exécutés simultanément, l'identifiant de texture de notre exemple n'est pas forcément le même sur tous les contextes. Dans ce cas, la texture ne s'affichera pas correctement, ou plus grave encore, se terminera brutalement. Le programmeur doit donc disposer d'un mécanisme d'attribution d'identifiant de textures transparent, quel que soit le nombre de contextes. L'introduction de la classe `ContextData` permet de pallier ce problème en garantissant que sur contexte le numéro de texture sera valide.

Le fonctionnement de ce mécanisme est relativement simple ; les données dépendant du contexte sont allouées en utilisant une classe *template* C++ *smart pointer* `ContextData`. En interne, chaque objet de ce type conserve une liste des données allouées pour chaque contexte. Lorsque l'application souhaite accéder à un numéro de texture, par exemple, le *smart pointer* regarde dans la liste des variables qu'il maintient à jour pour récupérer l'identifiant de texture propre au contexte courant.

Le mécanisme de données spécifique au contexte est assez simple à mettre en œuvre au niveau du programmeur, et nécessite peu de modifications dans son code initialement prévu pour tourner en environnement mono-contexte : seul le type de certaines variables change. La seule règle à respecter est d'initialiser et utiliser les variables lorsqu'un contexte est disponible ; en effet, lorsque le contexte est invalide[5], les données n'ont aucune existence.

4.4 Exemples d'utilisation

Dans cette section nous donnons deux exemples d'utilisation de la vrLib. Le premier est une application minimale dans laquelle nous affichons le message *Hello world !* La seconde application nous permet de montrer comment le programmeur peut indiquer, dans son code, que tel outil d'interaction doit agir sur tel objet.

Ces deux exemples ne sont pas très conséquents, leur code permet de montrer à quel point il est facile et simple d'écrire une application de réalité virtuelle avec la vrLib. Notons que le placement et l'orientation des objets sont automatiques dans le cas où le programmeur ne le précise pas. Dans les deux exemples qui suivent, les widgets sont placés au centre de la scène et orientés face à l'utilisateur. Au besoin, l'utilisateur peut utiliser les rayon virtuel pour attraper et déplacer les widgets librement dans l'environnement.

[5] Un contexte n'existe qu'à certains moments de l'application, notamment pour afficher des données.

4.4.1 Hello world !

L'application *Hello world !* est le premier programme d'exemple dans lequel on affiche un message. Il contient le strict minimum que le programmeur doit mettre en œuvre pour lancer une application de réalité virtuelle grâce à la vrLib (4.18).

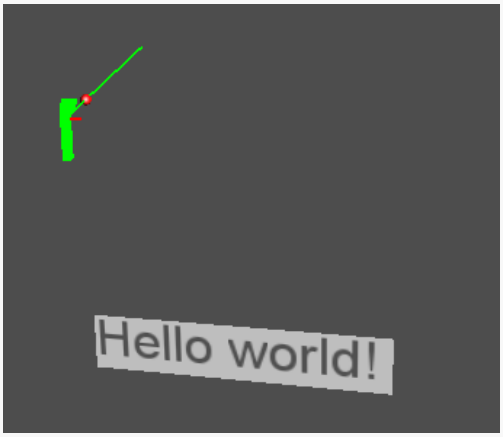


Fig. 4.18: Une application minimale avec la vrLib.

Nous indiquons ci-dessous le code source nécessaire à l'écriture de ce programme :

```
#include <vr/application.h>
#include <vr/cmp/wdg/label.h>

vr::Application *app;

int main(int argc, char **argv)
{
  app = vr::Application::create();
  vr::Label *label = new vr::Label("Hello world!");
  label->show();
  app->quit();
}

void draw(void)
{
  app->draw();
}
```

4.4.2 Changement d'outil d'interaction

Cette seconde application, très simple, permet de mettre en évidence la facilité de changement d'outil d'interaction par le programmeur. Dans cet exemple, nous souhaitons passer d'une technique par lancer de rayon classique à un pointeur de souris contraint dans le lieu géométrique d'un menu plat. Ainsi, lorsque l'utilisateur passe le rayon virtuel sur le menu, celui-ci se transforme en curseur $2D$ contraint, jusqu'à sortir du menu. La figure 4.19 montre l'application à deux instants différents : sur l'image de gauche l'outil est un rayon, et sur l'image de droite, l'outil est un curseur $2D$.

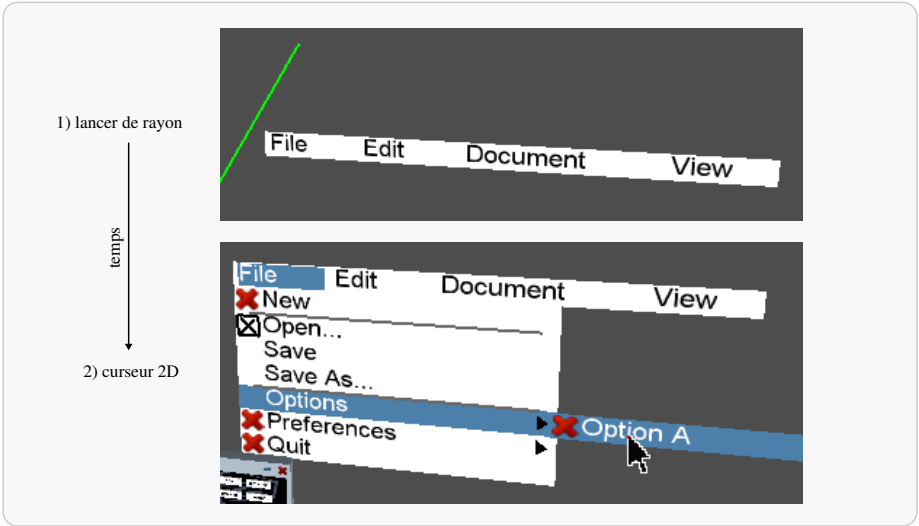


Fig. 4.19: Changement d'outil d'interaction avec la vrLib : le rayon est remplacé par un curseur $2D$ contraint dans le plan du menu, tant qu'il y reste.

Nous indiquons ci-dessous le code source nécessaire à l'écriture de ce programme :

```
#include <vr/application.h>
#include <vr/cmp/wdg/menu.h>
#include <vr/it/pointer2D.h>

vr::Application *app;

int main(int argc, char **argv)
{
  app = vr::Application::create();

  // création du menu File
  // avec plusieurs éléments
  vr::Menu *menu = new vr::Menu("File");
  menu->addItem("New");
```

```
   menu->addSeparator();
   menu->addItem("Open...")->setChecked(true);
   menu->addItem("Save");
   menu->addItem("Save As...");
   menu->addSeparator();
   menu->addMenu("Options")->addItem("Option  A");
   menu->addMenu("Preferences");
   menu->addItem("Quit");

   // création d'une barre de menu
   // qui contiendra les 4 menus
   vr::MenuBar *bar = new vr::MenuBar()
   bar->addMenu(menu);
   bar->addMenu("Edit");
   bar->addMenu("Document");
   bar->addMenu("View");
   bar->show();

   // par défaut l'outil est un rayon
   // nous le changeons en pointeur 2D pour ce menu
   menu->setInteractionTool(new vr::Pointer2D());

   app->quit();
 }

 void draw(void)
 {
   app->draw();
 }
```

4.4.3 Manipulation contrainte

Ce troisième exemple montre comment un objet de la vrLib peut être contraint par rapport à un autre objet. Nous souhaitons que l'utilisateur puisse déplacer une sphère sur la circonférence d'un cercle à l'aide de la technique du lancer de rayon. Pour cela, il suffit de définir un cercle d'un certain rayon, que l'on peut placer n'importe où dans la scène. Puis l'on définit une contrainte, sur la sphère par rapport au cercle, qui sera directement prise en charge par la vrLib. Les deux objets, cercle et sphère, se trouvent ainsi liés par le gestionnaire de contraintes et tout déplacement de la sphère par l'utilisateur répondra précisément à la contrainte définie. La figure 4.20 montre une sphère déplacée à par l'utilisateur.

Le code source pour cette application est donné ci-dessous.

```
#include <vr/application.h>
#include <vr/geom/circle.h>
#include <vr/geom/obj/sphere.h>
```

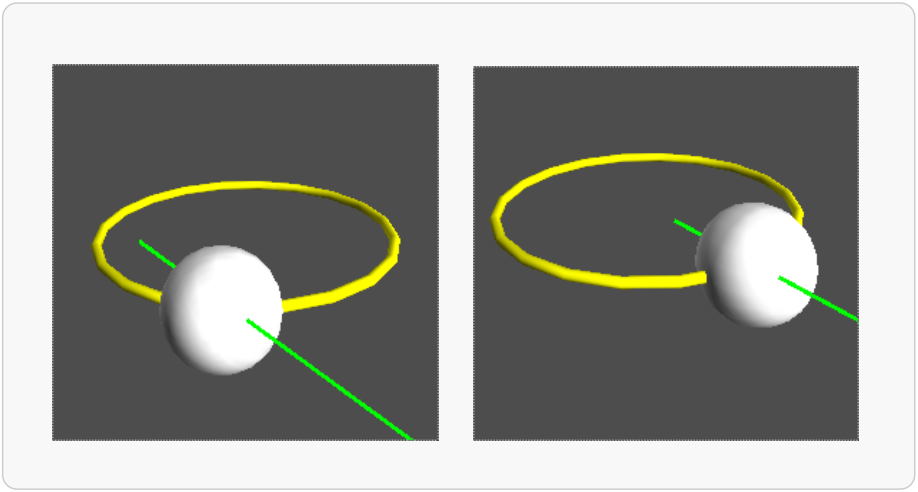


Fig. 4.20: Contrainte d'un sphère sur un cercle : le gestionnaire de contraintes fait le lien entre les deux objets.

```
vr::Application *app;

int main(int argc, char **argv)
{
  app = vr::Application::create();

  // creér un cercle de rayon 15 cm.
  vr::Circle *circle = new vr::Circle(0.15);

  // créer un objet sphère à contraindre
  vr::Sphere *sphere = new vr::Sphere();

  // ainsi que le cercle de contrainte
  // (centre, rayon, plan)
  sphere->setCenter(circle->getFrame());
  sphere->setRadius(circle->getRadius);
  sphere->setPlane(vr::CONS_PLANE_XZ);

  circle->show();
  sphere->show();
  app->quit();
}

void draw(void)
{
  app->draw();
}
```

4.5 Conclusion et perspectives

La vrLib a pour objectif d'aider le développeur a écrire rapidement et facilement des applications pour la réalité virtuelle. Ses 6 principales qualités sont de :

- permettre de développer facilement et rapidement une application fonctionnelle de manière similaire à ce que l'on fait en environnement $2D$;
- proposer un ensemble de widgets issus des environnements $2D$ pour le portage rapide d'applications $2D$ vers la $3D$;
- permettre de concevoir un environnement virtuel à l'aide de composants graphiques spécifiquement dédiés à la $3D$;
- proposer des outils d'interaction par défaut, et une interaction adaptative selon les besoins ;
- permettre de développer aisément des outils d'interaction qui serviront à l'utilisateur pour interagir avec son environnement virtuel ;
- offrir un cadre de développement solide et flexible selon les besoins du programmeur.

Il n'existe à l'heure actuelle aucune librairie dédiée à la réalité virtuelle qui permet d'écrire et de concevoir une application aussi simplement qu'en environnement de bureau. Tout développeur est soit confronté à une cohorte de librairies très compliquées et dans laquelle il finit par se perdre, soit réduit à re-développer selon ses propres besoins ce qui ne devrait pas constituer le cœur de son travail : l'interface et les outils d'interaction. Bien entendu, il n'existe pas d'interfaces génériques répondant à tous les usages et à toutes les applications, mais le programmeur doit avoir accès à un minimum d'outils génériques et simples d'utilisation pour la majorité de ses besoins. La vrLib offre justement la possibilité de porter assez simplement une application prévue pour la $2D$ en $3D$, puis d'utiliser des outils d'interaction propres au besoin de chaque application.

Il reste encore de nombreuses choses à réaliser sur cet outil de développement, avant qu'il n'arrive à maturité. La structure générale est maintenant établie, mais il reste à enrichir cette boîte à outils par des outils plus spécifiques à certains cas, de nouveaux objets spécifiques à la $3D$, afin de permettre plus de flexibilité dans les fichiers de configuration de l'interface, etc.

Nous voyons principalement deux perspectives à ce travail initial.

4.5.1 vrDesigner : un studio de programmation graphique

La première perspective qui pourrait suivre en parallèle le développement de vrLib est vrDesigner. Il s'agit d'un studio de programmation graphique qui permet au programmeur de concevoir une interface graphique en $3D$ directement à l'aide de l'environnement virtuel, comme il est possible de le faire en $2D$ avec des constructeurs d'interfaces du type Visual Studio et Qt Designer.

Le programmeur disposerait en premier les widgets dont il a besoin – fenêtres, boutons, menus, etc. –, à l'aide de la technique du glisser/déposer. Il serait également possible de définir ses propres widgets dans l'interface de vrDe-

signer. Ensuite, les signaux et les slots seraient connectés aux différents widgets à l'aide de cette même interface. Le chargement et la sauvegarde de l'interface du programme se feraient dans un fichier de description d'interface de type XML.

Une idée intéressante serait de pouvoir directement sélectionner l'outil d'interaction dédié par défaut à un widget de l'interface. Ainsi, lorsque l'utilisateur du programme passe l'outil d'interaction par défaut sur le widget en question, celui-ci serait remplacé par l'outil qui a été préalablement défini dans le constructeur d'interface jusqu'à ce qu'il ressorte du widget.

4.5.2 vrDesktop : un gestionnaire de fenêtres pour la réalité virtuelle

Il s'agit certainement de l'idée la plus ambitieuse et la plus longue. vrDesktop serait un gestionnaire de fenêtres – plus globalement de conteneurs $3D$ – permettant le lancement et l'utilisation d'applications directement depuis un environnement de travail virtuel de type Workench. Ce concept repose sur deux principes fondamentaux ; d'une part, le bureau virtuel tourne en permanence à l'instar de KDE ou Gnome en $2D$ et contient différents classeurs et icônes qui représentent les applications et les raccourcis aux fichiers et aux paramètres de l'environnement. D'autre part, les applications seraient entièrement manipulables dans cet environnement, soit en mode fenêtré, soit en mode plein écran.

Il est difficile d'inventer une nouvelle interface utilisateur, mais l'objectif serait d'offrir à l'utilisateur les mêmes possibilités d'interaction qu'il dispose dans le monde réel. vrDesktop agirait ainsi comme un environnement de travail intermédiaire entre un bureau $2D$ et un bureau réel et matériel. Il s'agit donc de pouvoir manipuler des icônes dans un univers de travail $3D$ contraint et simplifié visuellement de manière à ne pas perturber l'interaction. Ainsi, l'environnement permettrait de ranger, classer et empiler les documents de travail avec une grande liberté et une grande facilité. Un projet plus modeste mais du même genre a d'ores et déjà été initié pour un environnement de bureau par Agarawala et Balakrishan [AB06] : le BumpTop, qui propose une table de travail sur laquelle on empile les dossiers et les fichiers de travail que l'on sélectionne ensuite à l'aide d'un lasso de sélection.

Chapitre 5

Techniques d'interaction évoluées

5.1 Introduction

Le chapitre précédent nous a permis de voir comment écrire une application de réalité virtuelle à partir de la librairie vrLib. Celle-ci autorise soit le portage direct d'applications $2D$ vers un environnement immersif, soit l'écriture d'un logiciel avec une interface et des outils d'interaction, certes fonctionnels, mais pas nécessairement les plus adaptés à la réalité virtuelle.

Dès lors que nous sommes en mesure de concevoir une application de réalité virtuelle à l'interaction simple, nous souhaitons introduire des concepts d'interaction $3D$ originaux, propres à la réalité virtuelle. Ainsi, d'un premier programme à l'interaction assez simple, nous désirons arriver à un logiciel dont l'interaction $3D$ est totalement adaptée à un environnement de réalité virtuelle. Cela nécessite avant toute autre chose une approche de l'interaction novatrice ; nous pensons qu'il ne faut pas dissocier interaction $2D$ et interaction $3D$, mais au contraire intégrer toutes les dimensions de l'interaction dans un continuum.

En parallèle à cette démarche, nous avons investi le domaine très intéressant de l'interaction gestuelle. Nous cherchons à proposer un cadre de reconnaissance de postures et de gestes de manière à ce qu'un utilisateur puisse employer directement ses propres mains pour manipuler le monde virtuel qui l'entoure. Nous proposons deux techniques d'interaction innovantes principalement dédiées aux tâches de sélection et navigation, et pour lesquelles l'utilisation conjointe des deux mains est essentielle.

Nous présentons dans une première section notre approche de l'interaction conduite par la dimension, où nous expliquons notre modèle de conception des objets virtuels et des outils qui les manipulent. La seconde section illustre notre travail sur l'interaction gestuelle. Nous y présentons une méthode de calibrage simple, ainsi qu'un couplage de l'algorithme ICP avec le filtre de Kalman dans l'optique de stabiliser le suivi des mouvements des mains. Enfin, nous expliquons nos méthodes de reconnaissance des postures et des gestes. La troisième section présente le rayon déformable, une technique de sélection et de navigation souple manipulée à l'aide des deux mains. Enfin, la quatrième et dernière

section introduit deux méthodes de sélection multiple d'objets virtuels à l'aide de volumes.

5.2 Vers une interaction conduite par la dimension

Les contemporains de la réalité virtuelle, comme Bowman [Bow99], ont ouvert la voie en dégageant une taxonomie des techniques existantes, en les classant par groupes. Sélection, manipulation, navigation et contrôle du système sont des termes qui reviennent sans cesse. Pour autant, s'il est possible de classer une nouvelle métaphore sous l'une de ces appellations, aucune règle, aucune théorie n'existe pour aider à sa conception. Et il existe encore moins de plateforme de développement qui s'appuie sur cette base théorique pour déployer rapidement une application en environnement immersif.

Le développement rapide de la réalité virtuelle a permis de renforcer l'idée que l'interaction sur des objets de la scène est la difficulté majeure à résoudre. Ceux-ci sont placés dans un espace où ils peuvent à la fois être tournés et déplacés dans tous les sens. À l'évidence, ceux et celles qui pensaient que toutes ces entités graphiques devaient être traitées comme des objets $3D$ (*i.e.* possédant 6 degrés de liberté, 3 de rotation, et 3 de translation) ont fait fausse route. Seuls certains objets ont besoin d'être manipulés librement dans l'espace (ex : placer un oiseau virtuel dans le décor d'une scène). Mais, dans de nombreux cas, l'augmentation du nombre de degrés de liberté nuit à l'utilisateur (ex : placer un atome dans une molécule sans contraindre ses mouvements).

Charge cognitive élevée, degrés de liberté trop restreints ou trop importants, la liste des inconvénients à la démocratisation d'applications de réalité virtuelle est longue. Dès lors que l'homme a tenté de se rapprocher de la machine, il a essayé de rendre les interactions matérielles et logicielles les plus *transparentes* possibles. Les chercheurs se sont focalisés sur les tâches à résoudre à l'aide d'outils logiciels spécialisés pour leurs propres besoins applicatifs. Ceux qui se sont intéressés à proposer des outils d'interaction plus généraux ont vite déchanté : ils sont presque inutilisables car trop compliqués et inadaptés à la tâche qui leur est confiée.

Nous allons voir que les objets et les outils d'interaction possèdent deux dimensions : une dimension fonctionnelle et une dimension visuelle. Nous prenons le choix de ne nous consacrer, dans un premier temps, qu'aux aspects graphiques et à la visualisation des objets et des outils qui les manipulent. Précisons encore que nous ne considérons pas d'objets composites, tels que les humanoïdes qui sont des objets articulés.

Définition 1 La **dimension visuelle** est le nombre de dimensions nécessaires à l'affichage d'un objet graphique ou d'un outil d'interaction.

Définition 2 La **dimension fonctionnelle** est le nombre de degrés de liberté nécessaires au fonctionnement d'un objet graphique ou d'un outil d'interaction.

Dans une scène virtuelle, l'utilisateur agit sur les objets à l'aide de techniques d'interaction, par l'intermédiaire de périphériques physiques. La chaîne d'interaction est donnée par la figure 5.1.

Fig. 5.1: Schéma de la chaîne d'interaction entre l'utilisateur et l'application.

Pour illustrer ces notions, prenons par exemple le cas de l'utilisation du lancer de rayon sur un menu $2D$. Il s'agit d'une technique d'interaction dont la dimension visuelle est $1D$ et la dimension fonctionnelle est $6D - 3$ pour la rotation et 3 pour la translation. Est-ce l'outil le plus adapté à la manipulation d'un menu $2D$? Non puisque la charge cognitive nécessaire à l'orientation et au déplacement du rayon est plus élevée qu'avec un simple curseur $2D$ d'environnement de bureau. Il faut peut-être contraindre l'outil d'interaction en fonction de la nature de l'objet manipulé. Dans notre exemple, cela revient à utiliser un rayon qui ne s'oriente pas et ne se déplace que dans le plan du menu ; ceci revient à considérer le rayon comme un pointeur $2D$ contraint dans le lieu géométrique du menu $2D$. Nous nous servirons de cet exemple afin d'illustrer notre approche.

Dans un premier temps, nous allons introduire les notions de dimensions fonctionnelle et visuelle pour les objets virtuels. Dans un deuxième temps, nous faisons de même pour les outils d'interaction. Nous concentrons ensuite notre réflexion à l'établissement de règles de construction des objets et des outils pour que l'interaction logicielle devienne aussi transparente que possible pour l'utilisateur, c'est-à-dire intuitive, accessible et fonctionnelle. De ces règles nous donnons des lois à suivre dans le cas de la conception du couple outil - objet. Enfin, l'application des règles et des lois sera illustrée par un exemple simple.

5.2.1 Dimensionnalités des objets manipulés

Bien souvent, les concepteurs d'interfaces utilisateurs contemporains focalisent leurs efforts pour rendre les interfaces $3D$ visuellement attractives, comme si ce mot pouvait d'un coup de baguette magique transformer la citrouille en carrosse. Pourtant, faire tourner ces interfaces dans tous les sens aura certes un côté ludique, mais cela n'augmentera pas la productivité de l'utilisateur pour autant. La notion de productivité et de rendement est essentielle : créer une interface "jolie" et "design" n'a jamais réellement permis d'augmenter les performances de l'utilisateur. Certes, son confort est accru, et loin de nous l'idée de remettre en question les améliorations visuelles de ces dernières années. Mais il nous semble essentiel de souligner l'importance de ne pas se concentrer que sur ces aspects. Pour que les interfaces soient ergonomiques, il faut trouver le parfait équilibre entre qualités visuelles et qualités fonctionnelles des objets qui

sont manipulés.

En considérant le monde réel, on peut constater que tous les objets réels possèdent chacun ses propres dimensions visuelle et fonctionnelle. Lorsqu'on allume une lampe, le bouton sur lequel on appuie est de dimension fonctionnelle 1, le levier de vitesse de notre voiture est de dimension 2, etc. Bien que leur représentation visuelle et matérielle soit tridimensionnelle, la dimension fonctionnelle n'est pas toujours équivalente. Et c'est bien ici que se situe le cœur du problème. Il n'y a pas nécessairement d'égalité entre dimension visuelle et dimension fonctionnelle pour un objet "réel". Dans ce cas, pourquoi devrait-il y en avoir une dans le cas des objets "virtuels" ?

Certains concepteurs d'interfaces souhaitent remplacer, en environnement virtuel, les widgets $2D$ des interfaces de bureau par des widgets complètement $3D$. Or, après expérience, on peut constater que bien souvent, en ajoutant une ou plusieurs dimensions visuelles à un widget, les performances de l'utilisateur sur l'application se dégradent tandis que la charge cognitive associée augmente sensiblement. Le challenge est de trouver le meilleur équilibre entre la dimension visuelle et la dimension fonctionnelle du widget. Plus simplement, pour une tâche donnée, il faut s'interroger sur le nombre de dimensions fonctionnelles le plus adapté au travail qui est dévolu au widget. Quels sont ses degrés respectifs qui maximisent à la fois la productivité de l'utilisateur et son confort de travail ?

5.2.2 Dimensionnalités des outils d'interaction

Un équilibre doit être trouvé entre la dimension fonctionnelle et la dimension visuelle d'un objet virtuel. Dans la chaîne de l'interaction entre l'utilisateur et l'application, la place de l'outil est essentielle. En effet, si nos objets possèdent une certaine dimension fonctionnelle, pour que l'utilisateur puisse au mieux s'en servir, quelles sont les caractéristiques fonctionnelles et visuelles des outils qui sont les plus adaptées à ces objets ?

N'importe quel objet réel dispose d'une certaine dimension visuelle et fonctionnelle. Les outils qui permettent d'interagir avec de tels objets doivent être adaptés à la tâche qui leur est dévolue. Par exemple, il est possible d'enfoncer un clou sans marteau, mais cet outil est sans doute le plus adapté. De la même manière que les objets qu'ils sont chargés de manipuler, les outils définissent à la fois une dimension visuelle et une dimension fonctionnelle.

Dans un environnement virtuel, un utilisateur doit disposer de l'outil le plus adapté aux objets qu'il manipule. Il s'agit de trouver le meilleur équilibre entre la dimension visuelle et la dimension fonctionnelle de tout outil d'interaction. Quels sont ces degrés respectifs qui maximisent l'efficacité de manipulation et minimisent la charge cognitive du manipulateur ?

5.2.3 Conception de l'interaction : de l'objet à l'outil

Nous venons de discuter des dimensions fonctionnelles et visuelles des entités virtuelles qui font le lien entre d'une part l'utilisateur et le périphérique qu'il tient en main, et, d'autre part, la machine qui crée l'environnement virtuel : les objets manipulés et les outils d'interaction. Nous proposons 4 étapes à mettre en œuvre lors de la phase de conception de l'interaction :

1. établir la dimension fonctionnelle de l'objet ;
2. établir la dimension fonctionnelle de l'interaction en fonction de celle de l'objet ;
3. établir la dimension visuelle de l'objet ;
4. établir la dimension visuelle de l'outil d'interaction, en fonction de la dimension visuelle de l'objet.

Nous pouvons voir que notre approche consiste à concevoir l'objet en premier, puis l'outil en fonction de l'objet considéré. Il nous semble en effet important de souligner que c'est l'outil qui doit être adapté à l'objet et non l'inverse. Nous pouvons illustrer ce processus de conception à l'aide d'un exemple.

exemple du menu $2D$ L'utilisateur souhaite par exemple interagir avec un menu $2D$. Déterminons les dimensions fonctionnelles et visuelles de l'objet menu et de l'outil qui permet de le manipuler :

1. dimension fonctionnelle d'un menu : $2D$ dans le plan XY du menu ; en effet, il faut :
   - 1 dimension pour naviguer verticalement dans la liste d'éléments ;
   - 1 dimension pour naviguer horizontalement dans un sous-menu ;
2. dimension fonctionnelle de l'outil : $2D$ dans le plan XY du menu, pour accéder aux éléments du menu et aux sous-menus ;
3. dimension visuelle du menu : légère épaisseur $3D$ pour perception du menu en vue de profil ;
4. dimension visuelle de l'outil : un pointeur $2D$[1] suffit.

Nous pouvons ensuite apporter quelques améliorations à notre couple menu-pointeur. L'outil d'interaction est virtuel et son déplacement est donné par celui d'un périphérique physique, en ne conservant que deux dimensions de déplacement. Il est possible de diminuer les erreurs de sélection en imposant que le pointeur passe d'une entrée à une autre par un saut, et non de manière continue. Bien que le menu puisse s'orienter de façon quelconque dans l'environnement, il pourrait être réorienté en face de l'utilisateur le temps de sa manipulation, puis revenir à sa position initiale. Enfin, nous proposons d'ajouter un certain niveau de transparence, pour que l'utilisateur puisse voir le reste de la scène.

Nous venons de voir 4 étapes successives dédiées à la conception d'un couple outil d'interaction - objet à manipuler. La section suivante donne les 3 lois de

[1] nous entendons un curseur contraint dans le plan du menu.

l'interaction qui servent à fixer le nombre minimal de dimensions fonctionnelles de l'outil et de l'objet.

5.2.4 Les 3 lois de l'interaction

Nous souhaitons diminuer autant que possible le nombre d'intermédiaires entre l'utilisateur et l'application sur laquelle il travaille, en proposant une interaction adaptée à la tâche. Nous proposons 3 lois à mettre en œuvre pour que tout outil soit adapté aux objets qu'il manipule, et que l'interaction soit la plus efficace et transparente possible.

loi 1 La dimension fonctionnelle du périphérique doit être supérieure ou égale à celle de l'objet.

loi 2 La dimension fonctionnelle de l'objet doit être minimale, de manière à conserver toutes ses fonctionnalités.

loi 3 La dimension fonctionnelle de l'outil d'interaction doit être réduite à la dimension fonctionnelle de l'objet.

La première loi permet de s'assurer que le périphérique pourra effectivement permettre de manipuler dans de bonnes conditions l'objet à manipuler. Par exemple, une souris $2D$ est adaptée pour manipuler un menu $2D$; cependant, un périphérique $6D$ comme un Flystick peut également servir à le manipuler convenablement.

La seconde loi sert à garantir que l'objet se limite bien au travail qui lui est confié, sans rien omettre de la tâche qui lui est impartie. Par exemple, un menu offre un accès à une arborescence d'éléments, il n'y a donc aucun intérêt à lui adjoindre une dimension supplémentaire.

La troisième et dernière loi permet de s'assurer que l'outil est bien adapté à l'objet qu'il doit manipuler. Ainsi, un périphérique d'interaction dont la dimension fonctionnelle est supérieure à celle de l'objet, pourra convenir à la manipulation de l'objet. Par exemple, un menu de dimension fonctionnelle 2 peut être manipulé à l'aide du Flystick d'ART à 6 degrés de liberté. Cependant, l'outil d'interaction ne prend en considération que les déplacements dans le plan XY du périphérique.

Soit D[2] un périphérique d'interaction, T[3] un outil d'interaction, et O[4] l'objet à manipuler.

Connaissant $d_f(D)$, la dimension fonctionnelle du périphérique d'interaction, et $d_f(O)$ la dimension fonctionnelle de l'objet à manipuler, nous souhaitons déterminer la dimension fonctionnelle $d_f(T)$ de l'outil d'interaction. D'après les 3

[2]a. *Device*
[3]a. *Tool*
[4]a. *Object*

lois précédentes, nous pouvons alors écrire :

Si $d_f(D) \geqslant d_f(O)$, alors $d_f(T) = d_f(O)$.

exemple Nous souhaitons déterminer une meilleure interaction possible pour allumer la lumière dans une scène virtuelle.

Nature du périphérique : un Flystick à 6 degrés de liberté.

Nature de l'objet :
- a. 1 bouton "allumé / éteint" – 2 états pour une lumière standard ;
- b. 1 valuateur de puissance – n états compris entre 2 bornes pour une lumière à intensité variable.

Dimension fonctionnelle de l'objet :
- a. $1D$ 0 ;1, car 2 états. D'après la loi 1, on a bien $d_f(D) \geqslant d_f(O)$;
- b. $1D$ [0...1], car n états. D'après la loi 1, on a bien $d_f(D) \geqslant d_f(O)$.

Dimension fonctionnelle de l'outil d'interaction pour qu'il puisse actionner l'interrupteur :
- a. $d_f(T) = d_f(O) = 1D$, d'après la loi 3 ;
- b. $d_f(T) = d_f(O) = 1D$, d'après la loi 3.

Visuellement, il faut au moins que la dimension visuelle de l'outil soit supérieure ou égale à sa dimension fonctionnelle, et il en est de même l'entité. Ceci afin que la dimension fonctionnelle de l'outil et de l'entité soit compréhensible par l'utilisateur.

Dimension visuelle de l'entité :
- a. $2D$ contraint, par exemple un bouton pressoir plat ;
- b. $2D$ contraint, par exemple une barre coulissante plane.

Dimension visuelle de l'outil d'interaction :
- a. $3D$ contraint, par exemple l'avatar d'un doigt qui se colle à l'interrupteur et l'enfonce ;
- b. $3D$ contraint, par exemple l'avatar d'un doigt qui se colle à la barre coulissante pour faire varier l'intensité lumineuse.

5.2.5 Conclusion

Nous venons de voir qu'il est possible de mettre en œuvre des règles et des lois pour l'interaction homme – application. Notre apport se situe essentiellement dans la manière originale de considérer la chaîne d'interaction entre l'utilisateur et le programme. Notre approche ouvre des perspectives de recherches intéressantes.

Multi-sensorialité Nous venons d'envisager une réponse au problème de la définition concrète des entités graphiques d'un environnement de Réalité Virtuelle et des outils d'interaction qui agiront dessus. Ici, nous avons considéré

uniquement le sens de la vue, mais il est également possible d'étendre cette théorie aux autres sens de l'utilisateur – le son et l'ouïe, le toucher et le tactile, etc.

Interaction adaptative La problématique de l'interaction générale questionne l'accès à un ensemble d'outils adaptés à l'ensemble des objets présents dans la scène. La question devient alors : sachant que la scène est composée d'objets de dimensionnalité fonctionnelle différente, comment faire pour que l'interaction soit transparente pour l'utilisateur ?

La réponse est assez simple : l'interaction doit s'adapter à la dimension fonctionnelle des objets de la scène. Tout comme un menuisier aurait besoin successivement d'un tournevis pour une vis, puis d'un marteau pour un clou, ici l'outil s'adaptera automatiquement à l'objet en fonction de sa dimensionnalité fonctionnelle. On pourra par exemple avoir un pointeur $2D$ pour des menus $2D$, un rayon laser pour les déplacements libres dans l'espace, etc.

En pratique, la vrLib intègre d'ores et déjà un tel mécanisme de changement d'outil à la volée. Soit le programmeur décide que tel objet nécessite tel paradigme au moment où il écrit son application, soit la librairie fournit automatiquement un outil par défaut.

En sus, on pourrait également imaginer une interaction adaptative selon d'autres critères. Par exemple, selon que les objets de la scène se situent loin ou non, l'outil pourrait passer du statut "rayon" au statut "rayon à ouverture".

5.3 Interaction gestuelle

Dans la vie de tous les jours, nos mains constituent notre principal outil d'interaction avec le monde réel. Elles nous servent à tenir un verre, faire un signe de la main, appuyer visuellement un discours, etc. En réalité virtuelle le suivi des mouvements du corps peut permettre d'interagir de manière très naturelle avec les objets de la scène. Nous souhaitons donc reconnaître des postures et des gestes à partir des valeurs renvoyées par les gants de données que nous avons à notre disposition. Ceux-ci incorporent également un mécanisme de suivi des mains dans l'espace – par exemple *via* des capteurs infrarouges. Il faut donc s'assurer de la validité des informations renvoyées par les gants, et par le système de suivi. Puis mettre en place un système qui autorise la reconnaissance des postures et des gestes.

Le matériel que nous utilisons retourne des valeurs plus ou moins bruitées. Les deux principaux problèmes sont donc de :

- corriger les valeurs de suivi ;
- calibrer correctement les valeurs d'angles de flexion des doigts.

Les valeurs d'angles n'ont pas besoin d'être corrigées ; en effet, nos tests préalables sur différents gants de données ont montré que les valeurs sont soit

complètement aberrantes – il est donc impossible de les corriger, soit suffisamment cohérentes pour être utilisées. Cependant, l'échelle de variation des valeurs n'est pas la même entre différents utilisateurs. En effet, leur morphologie palmaire peut varier sensiblement, un calibrage est donc nécessaire.

Lorsque les valeurs des capteurs ont été calibrées et corrigées, nous désirons ajouter un mécanisme de reconnaissance de postures et de gestes afin de les associer à des commandes de l'utilisateur. La reconnaissance de postures revient à déterminer des configurations fixes de la main qui peut éventuellement bouger dans l'espace, comme par exemple le poing fermé. Inversement, la reconnaissance de gestes associe une sémantique à l'évolution des postures au cours du temps.

Notre objectif est double : il s'agit, d'une part, de corriger et calibrer les valeurs des capteurs retournées par le matériel, et ,d'autre part, de reconnaître correctement les postures et gestes de l'utilisateur. Cette section présente dans une première partie un algorithme d'appariement itératif de points[5], associé à un filtre de Kalman. Il s'agit à la fois d'assurer la robustesse des valeurs retournées par les capteurs de suivi de la main dans l'espace, et aussi de rendre les mouvements de la main consistants dans le temps ; puis, dans une deuxième partie, nous voyons comment calibrer les valeurs d'angles de flexion des doigts. Ensuite, nous montrons comment fonctionne notre méthode de reconnaissance de postures, grâce notamment à un réseau de neurones bien construit, avant d'expliquer notre vision de la reconnaissance des gestes à l'aide d'un langage qui décrit l'évolution posturale et spatiale des mains.

5.3.1 Correction du suivi de mouvement

Les valeurs de suivi retournées par les capteurs sont bruitées, et parfois même complètement incohérentes. Par exemple, lorsque l'utilisateur ne bouge pas sa main – on parle alors de stabilité statique, des sauts dans l'orientation et la position apparaissent avec certains systèmes de suivi. Parmi ceux que nous avons à notre disposition et qui présentent de tels problèmes, on trouve des dispositifs infrarouges. Leur fonctionnement est assez simple : une structure rigide de capteurs infrarouges – éventuellement des émetteurs – est positionnée de manière fixe sur la main. Les informations enregistrées par l'ordinateur correspondent à la position des capteurs dans l'espace.

ICP nous permet de réduire les incohérences locales de position et d'orientation par la prise en compte de mauvais capteurs. Cependant, dans le temps, des sauts peuvent apparaître et rendre le mouvement irrégulier et illogique. Nous ajoutons un filtre de Kalman [Kal60] pour diminuer ces artefacts et lisser, dans une certaine mesure, les déplacements de la main. Nous voyons premièrement notre algorithme ICP modifié, et deuxièmement l'application du filtre de Kalman sur les données issues d'ICP.

[5] a. *Iterative Closest Points* ou *ICP*

Algorithme ICP

L'algorithme ICP pour Iterative Closest Points, introduit par [CM92] et [BM92], est utilisé principalement pour permettre le repositionnement de maillages lors de l'acquisition d'objets $3D$ à l'aide d'un système optique. Lorsque le système optique effectue plusieurs balayages pour parcourir l'objet intégralement, il se peut qu'il y ait des décalages entre les acquisitions. Cette méthode permet de faire correspondre les balayages entre eux afin de restituer l'objet dans son intégralité.

Notre version d'ICP se base sur la géométrie de la structure rigide théorique des capteurs qui est supposée connue. Il s'agit d'abord de rejeter tous les capteurs erronés, puis de minimiser l'erreur métrique sur l'ensemble de ceux qui sont valides, c'est-à-dire ceux qui sont visibles pour le système de suivi. Le but est de conserver les capteurs qui se rapprochent le plus de la structure théorique ; c'est ce que nous appelons la stabilité structurelle. On peut résumer cet algorithme aux six étapes suivantes :

- Sélectionner un ensemble de points dans un ou deux maillages.
- Établir une correspondance entre les points d'un maillage avec ceux de l'autre.
- Pondérer la correspondance de manière appropriée.
- Rejeter certaines paires en les traitant individuellement ou en considérant l'ensemble des paires.
- Calculer l'erreur métrique en se basant sur les paires obtenues.
- Minimiser l'erreur métrique.

L'article de Rusinkiewicz et Levoy [RL01] pourra apporter plus de précisions sur les différentes variantes de l'algorithme qui ont déjà été testées avec des résultats plus ou moins satisfaisants.

Le cas particulier que nous allons traiter est l'application de l'algorithme ICP à la capture de la translation et de la rotation de la main d'un utilisateur. L'intérêt que nous avons à mettre en œuvre une telle méthode vient du fait que nous souhaitons utiliser des gants de données. Ces périphériques possèdent des capteurs de position qui permettent de suivre les mouvements de la main dans l'espace. Les capteurs de positions peuvent, pour une raison purement matérielle ou physique – les capteurs peuvent être cachés par un autre objet, retourner des valeurs erronées ou en nombre insuffisant, ce qui entraîne un mouvement imprécis et instable.

La connaissance du modèle théorique de la position des capteurs nous permet d'identifier les capteurs qui semblent les plus cohérents et d'éliminer les données bruitées. Ceci simplifie l'algorithme ICP initial puisque contrairement au cas de l'acquisition d'objets $3D$ nous sommes en mesure d'identifier les paires théoriques/réelles automatiquement donc la deuxième étape de l'algorithme n'aura pas lieu d'être. De plus, ICP permet de donner des résultats en temps réel, ce qui est essentiel dans le cadre d'une interaction homme-machine. Nous résumerons donc notre version aux étapes suivantes :

– sélection des capteurs les plus cohérents selon notre heuristique ;
– rejet des capteurs faussés ;
– calcul de l'erreur métrique de la transformation basé sur la méthode de Matabosh *et al.* [MFS05] à partir d'un triplet de capteurs ;
– minimisation de l'erreur métrique en itérant sur tous les autres triplets de capteurs préalablement retenus.

Il est possible que trop peu de capteurs soient visibles ou que les données enregistrées sur les capteurs ne soient pas suffisamment cohérentes – bruit trop important – pour permettre le calcul de la transformation de manière précise. Dans ce dernier cas, le filtre de Kalman permettra de toute façon d'extrapoler cette transformation.

Heuristique de sélection des capteurs Nous connaissons la distance de tous les capteurs entre eux sur le modèle théorique. Nous calculons à chaque itération cette distance sur les valeurs réelles retournées par le matériel. Lorsque la distance réelle est supérieure à la distance théorique, d'un pourcentage Δ, pour tous les couples possibles, alors le capteur qui introduit trop d'erreurs est automatiquement rejeté. En effet, nous pouvons considérer qu'il est trop loin du modèle théorique, comme le montre l'exemple de la figure 5.2. Dans le cas où deux capteurs – ou plus – se retrouvent isolés, aucune information ne nous permet de savoir si ce sont ces capteurs, qui semblent correctement placé entre eux, qui sont valides ou les autres.

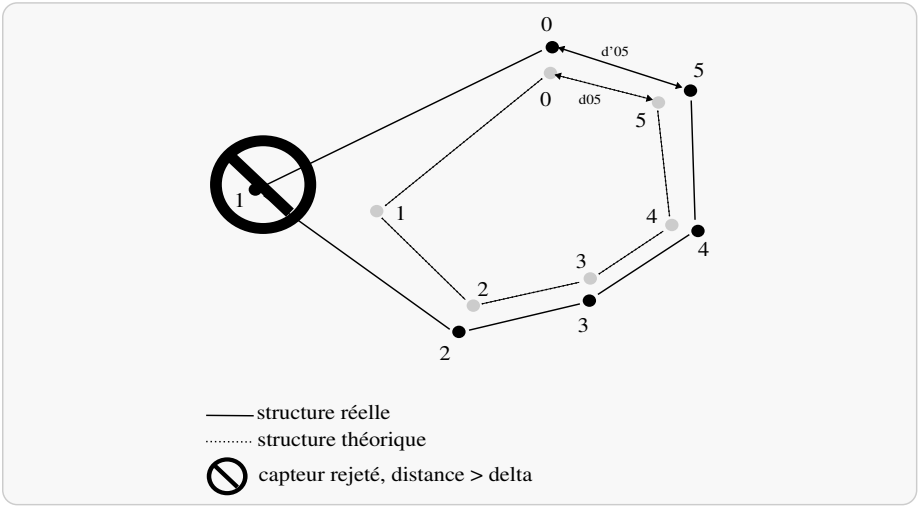


Fig. 5.2: Heuristique d'élimination des capteurs en fonction de leur distance relative.

Définition formelle Donnons maintenant une version plus formelle de cet algorithme.

Soit :

- A_i tel que $i \in \{1, 2, \ldots, n\}$ les coordonnées du capteur théorique i du gant centré au niveau de l'origine du repère global O ;
- B_i tel que $i \in \{1, 2, \ldots, n\}$ les coordonnées du capteur i situé sur le gant sans présence de bruit ;
- C_i tel que $i \in \{1, 2, \ldots, n\}$ les coordonnées du capteur i situé sur le gant avec présence de bruit aléatoire.

Le but est de s'approcher de la transformation T qui transforme A_i en B_i. On définit $T = (R, t)$ tel que R représente la rotation, et t la translation avec $R \wedge t \in \mathbb{R}$. On exprime cette transformation à l'aide d'une matrice de la forme :

$$T = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_{14} \\ r_{21} & r_{22} & r_{23} & t_{24} \\ r_{31} & r_{32} & r_{33} & t_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

On procède d'abord à la phase d'élimination des capteurs qui sont trop éloignés du modèle théorique, à l'aide de notre heuristique de distance. Si pour tout capteur $i \wedge j \in \{1, 2, \ldots, n\}$ on a $\frac{d'_{ij} - d_{ij}}{d_{ij}} > \Delta$, et ceci $n - 1$ fois alors le capteur est éliminé de la sélection.

Le calcul de la transformation s'effectue par la sélection d'un triplet de points observés $\{c_1, c_2, c_3\}$ auquel on ajoute le point c_4 – c_4 est le point qui correspond à l'extrémité de la normale du plan formé par le triplet. De la même manière, on obtient le quadruplet $\{a_1, a_2, a_3, a_4\}$ associé au modèle théorique.

La matrice de transformation se calcule donc par résolution du système d'équations suivant, à l'aide d'une méthode de Gauss-Jordan :

$$\begin{pmatrix} c_{1x} \\ c_{1y} \\ c_{1z} \\ \vdots \\ c_{4x} \\ c_{4y} \\ c_{4z} \end{pmatrix} = \begin{pmatrix} a_{1x} & a_{1y} & a_{1z} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{1x} & a_{1y} & a_{1z} & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{1x} & a_{1y} & a_{1z} & 1 \\ & & & & & \vdots & & & & & & \\ a_{4x} & a_{4y} & a_{4z} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{4x} & a_{4y} & a_{4z} & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{4x} & a_{4y} & a_{4z} & 1 \end{pmatrix} \begin{pmatrix} r_{11} \\ r_{12} \\ r_{13} \\ t_{14} \\ \vdots \\ r_{33} \\ t_{34} \end{pmatrix}$$

On a donc $T(A_i) + e = B_i$ avec $e \to 0$ et B_i étant inconnue. Avec $e = \frac{1}{n} * \sum_{i=1}^{n} \|C_i - T * A_i\|^2$ l'erreur à minimiser. On conserve alors la transformation T tel que e soit minimum.

Algorithme Nous venons de voir la définition formelle de notre algorithme ICP. La figure 5.3 en donne le déroulement.

Résultats L'introduction du filtre ICP permet de corriger assez sensiblement les mesures issues des capteurs d'un gant de données. Le gant est placé dans l'espace sur une structure métallique dont la position est connue. Nous disposons donc de trois valeurs que nous comparons :

1 *Coordonnées des capteurs théoriques*
Données : $a[n]$: Liste de coordonnées
2 *Coordonnées des capteurs réels*
Données : $c[n]$: Liste de coordonnées
3 *Nombre d'erreurs enregistrées*
Données : $error[n]$: Liste de coordonnées
4 *Transformations calculées*
Données : $T[n]$: Tableau de matrices
5 *Erreur métrique par rapport à la transformation*
Données : $errmet[n]$: Tableau de réels
6 *Transformation utilisée*
Données : $Tfinal$: Matrice
7 *Pourcentage d'erreur de distance maximale tolérée*
Données : Δ : Réel
8 *À chaque rafraîchissement du gant*
9 **pour** $i \leftarrow 1$ **à** n **faire**
10 | $error[i] = 0$;
11 **fin Pour**
12 *Élimination des capteurs isolés*
13 **pour** $i \leftarrow 1$ **à** n **faire**
14 | **pour** $j \leftarrow i+1$ **à** n **faire**
15 | | **si** $\frac{distance(c[i],c[j]) - distance(a[i],a[j])}{distance(a[i],a[j])} > \Delta$ **alors**
16 | | | $error[i] = error[i] + 1$;
17 | | | $error[j] = error[j] + 1$;
18 | | **fin Si**
19 | **fin Pour**
20 **fin Pour**
21 **pour** $i \leftarrow 1$ **à** n **faire**
22 | **si** $error[i] == (n-1)$ **alors**
23 | | `elimine(`$c[i]$`);`
24 | **fin Si**
25 **fin Pour**
26 *Pour tous les triplets, on calcule la transformation et l'erreur correspondante*
27 **pour** $i \leftarrow 1$ **à** n **faire**
28 | **pour** $j \leftarrow i+1$ **à** n **faire**
29 | | **pour** $k \leftarrow j+1$ **à** n **faire**
30 | | | $T[i+j+k-6]$ =`GaussJordan(`$a[i], a[j], a[k], c[i], c[j], c[k]$`);`
31 | | | errmet$[i+j+k-6] = \frac{1}{N_p} * \sum_{i=1}^{N_p} \|C_i - T * A_i\|^2$;
32 | | **fin Pour**
33 | **fin Pour**
34 **fin Pour**
35 *On conserve la transformation qui minimise l'erreur*
36 *indice* =`rechercheIndiceErreurMin(`*errmet*`);`
37 Tfinal= $T[indice]$;

FIG. 5.3 : Algorithme ICP modifié.

– position du gant idéale ;
– position du gant sans filtre ;
– position du gant avec filtre.

Le tableau 5.3.1 donne l'écart type mesuré sur 1000 valeurs enregistrées. L'écart type moyen sans filtre est de 6,5 mm, alors qu'avec notre version d'ICP, l'écart type tombe à 2,4 mm. Nous pouvons voir deux choses : d'une part le filtre ICP corrige la position en moyenne d'un ratio de 2,72. D'autre part, les valeurs en X et en Z sont mieux corrigées que celles en Y. La figure 5.4 montre l'ensemble des mesures réalisées sans filtre, par rapport à la position idéale et celles avec filtre ICP.

	Écart type sans filtre	Écart type avec filtre	Ratio
X	0,001632	0,000558	2,923178
Y	0,001708	0,000966	1,768000
Z	0,003182	0,000870	3,657442
Total	0,006522	0,002394	2,724310

Tab. 5.1: Amélioration des mesures avec filtre ICP (sur 1000 valeurs, gant P5, en mètre).

Il apparaît que l'application du filtre ICP permet de mieux repérer le gant de données P5 dans l'espace. Notre version de l'algorithme offre la possibilité de mieux placer la structure théorique par rapport aux valeurs retournées par les capteurs. En réduisant les erreurs de repérage du gant dans l'espace, nous augmentons la précision des mouvements de la main. Un second algorithme est utilisé sur les données corrigées pour stabiliser les mouvements dans le temps et l'espace : le filtre de Kalman.

Filtre de Kalman

Le filtre de Kalman a été développé par Rudolph Kalman dans les années 1960 [Kal60] dans le cadre du programme spatial Apollo, et s'avère très utile dans de nombreux domaines : de la navigation – spatiale, terrestre et marine – à l'entraînement de réseaux de neurones. Il est utilisé principalement pour estimer l'état d'un système dynamique à partir d'une série de mesures bruitées ou incomplètes. Il s'agit d'un outil d'estimation de variables qui permet la minimisation de la variance de l'erreur estimée, et qui nous semble bien adapté à notre problème.

Dans le cas du suivi de mouvements, il peut être intéressant d'utiliser un tel filtre pour minimiser l'impact du bruit dans la détection de la position et de l'orientation, comme par exemple les sauts de valeurs dûs à la précision des capteurs utilisés sur des gants de données. Le filtre de Kalman fait appel à la dynamique de la cible – ici, la main de l'utilisateur – qui définit son évolution dans le temps pour obtenir des données cohérentes, en éliminant l'effet du bruit. Dans notre cas, nous souhaitons prédire les mouvements des mains de l'utilisateur en introduisant un léger décalage de l'ordre de 5 valeurs dans la détection des positions.

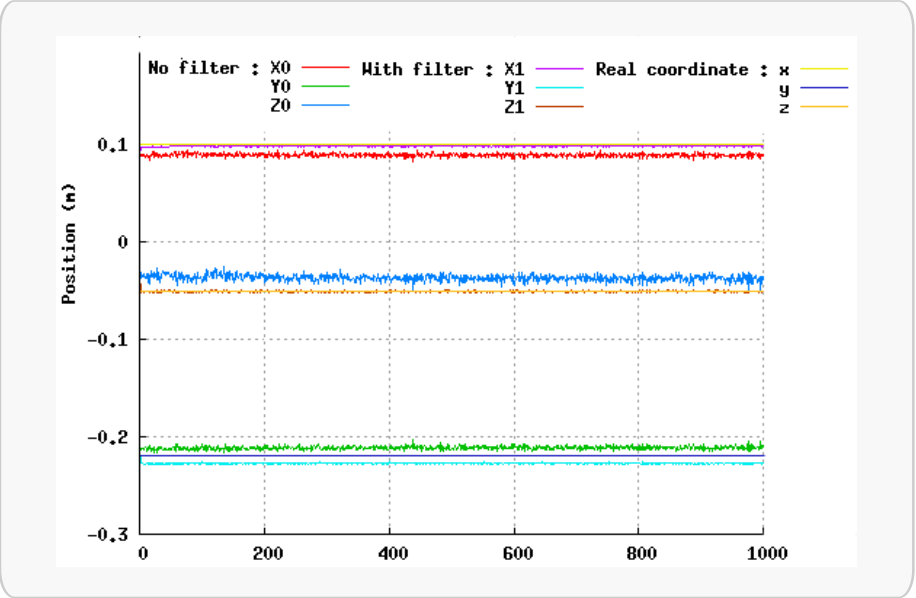


Fig. 5.4: Comparaison des valeurs corrigées avec ICP par rapport aux valeurs idéales et mesurées sans filtre.

Les étapes principales de notre algorithme sont les suivantes :
- filtrage des positions effectuant de trop grands sauts par rapport aux mesures précédentes pour améliorer la précision de l'algorithme ;
- calcul des matrices de prédiction ;
- estimation des nouvelles valeurs.

Définition formelle Donnons maintenant une version plus formelle de ce filtre.

Soit :

- k : l'index de temps ;
- x_k : l'état réel du système au temps k ;
- y_k : l'état mesuré à l'instant k ;
- w_k : le bruit à l'instant k – fluctuation parasite dans le mouvement ;
- z_k : le bruit de mesure à l'instant k ;
- v_k : la vitesse à l'instant k ;
- p_k : la position à l'instant k ;
- u_k : l'accélération à l'instant k ;
- T : l'intervalle de temps entre deux mesures.

Selon le modèle physique d'une main en déplacement nous pouvons décrire l'état x_{k+1} en fonction de x_k et de v_k. Nous définissons que l'état du système à l'instant k consiste en une position et en une vitesse, soit : $x_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix}$, avec

$$\begin{cases} p_k = x_{k-1} + Tv_{k-1} + \frac{T^2}{2}u_{k-1} + w_{k-1} \\ v_k = v_{k-1} + Tu_{k-1} + w_{k-1} \end{cases}$$

En définissant les matrices suivantes :

$$A = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}, B = \begin{bmatrix} T^2/2 \\ T \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Le système linéaire peut donc être résumé par le système d'équations suivant :

$$\begin{cases} x_{k+1} = Ax_k + Bu_k + w_k \\ y_k = Cx_k + z_k \end{cases}$$

Afin de résoudre ce système nous avons besoin d'une estimation précise de la position courante p et de la vitesse courante v.

Le filtre de Kalman nécessite qu'il n'y ait pas de corrélation entre le bruit dans le mouvement – w – et le bruit provoqué par les capteurs – z –, et que ces deux valeurs soient les plus proches de zéro. Ce que nous assurons par le fait d'éliminer les valeurs effectuant un trop grand saut et en appliquant préalablement notre algorithme ICP.

Nous considérons donc les deux variances indépendantes suivantes :

– variance de bruit dans le mouvement $S_w = w_k w_k^T$;
– variance de bruit dans la mesure $S_z = z_k z_k^T$.
Telles que w_k et z_k correspondent aux variances attendues.

On calcule la matrice P qui correspond à la variance de l'estimation de l'erreur :

$P_{k+1} = AP_kA^T + S_w - AP_kC_z^{-1}CP_kA^T$
On peut alors calculer la matrice K du gain de Kalman :

$K_k = AP_kC^T(CP_kC^T + S_z)^{-1}$
La matrice du gain nous permet d'estimer la valeur de x_{k+1}.
$\hat{x}_{k+1} = (A\hat{x}_k + Bu_k) + K_k(y_{k+1} - C\hat{x}_k)$
Le premier terme de cette équation donne l'état au temps $k+1$ de A fois l'estimation au temps k additionné de B fois l'accélération. Le deuxième terme de l'équation corrige l'estimation à partir de la mesure effectuée au même instant.

Algorithme

On considère une vitesse constante :
On est dans le cas d'un système linéaire avec une accélération nulle.
Valeur limite acceptée entre deux mesures de position
Données : β : Réel
Variance du bruit du mouvement
Données : M : Réel
Variance du bruit de la capture
Données : N : Réel
Ancienne valeur de position
Données : *anciennepos* : Coordonnées
Index de temps
Données : k : Temps
Coordonnées du gant
Données : *pos* : Coordonnées
Matrice de variance du bruit du mouvement
Données : S_w : Matrice
Matrice de variance du bruit de capture
Données : Sz : Matrice
Gain de Kalman
Données : K : Matrice
Variance de l'estimation de l'erreur
Données : P : Matrice
Matrice d'isolement des coordonnées dans les matrice 6x6
Données : C : Matrice

Filtre des sauts
si `distance`(*pos,anciennepos*)$> \beta$ **alors**
  $pos = anciennepos$;
fin Si
sinon
  $anciennepos = pos$;
fin Si

Calcul des matrices
S_w= `matriceIdentite`(*6,6*);
$S_w = S_w = S_w * M^2$;
S_z= `matriceIdentite`(*6,6*);
$S_z = S_z * N^2$;
matrice 6x6 adapté à : x, v_x, y, v_y, z, v_z $A =$ `matriceIdentite`(*6,6*);
on assigne le temps k tel que défini dans la définition formelle – T – $A[0,1] = A[2,3] = A[4,5] = k$;
$C =$ `matriceNulle`(*3,6*);
$C[0][0] = C[1,2] = C[2,4] = 1$;

Calcul du gain de Kalman
$K_k = AP_kC^T(CP_kC^T + S_z)^{-1}$;

mise à jour de l'estimation de la variance d'erreur
$P = APA^T + S_w - AP_kC^TS_z^{-1}CP_KA^T$;

Prédiction
$pos = A * anciennepos + P_kC^T(CP_kC^T + S_z)^{-1}$;

FIG. 5.5 : Algorithme Kalman.

Résultats Nous désirons mesurer l'impact de l'introduction du filtre de Kalman pour la prédiction des positions de la main de l'utilisateur. Il est cependant difficile de réaliser un mouvement linéaire parfait ; nous ne pouvons donc mesurer avec précision l'erreur réelle. Nous ne pouvons donc donner qu'une estimation visuelle de la correction. La figure 5.6 montre la trajectoire réelle et corrigée d'une main selon un mouvement circulaire dans l'espace.

On peut voir sur cette figure que le filtre de Kalman introduit une certaine latence qui s'explique par une précision plus ou moins importante de la mesure de la vélocité. Ce n'est pas particulièrement génant, tant que les mouvements de la main ne sont pas trop rapides. On observe cependant que l'application du filtre permet le rattrapage des sauts d'erreurs tout en conservant un suivi de mouvement satisfaisant. Kalman fonctionne mieux quand la présence de bruit sur les mesures de position est faible. On comprend donc que l'association avec ICP donne de meilleurs résultats, puisque cet algorithme diminue les erreurs de mesure. Là encore on ne peut mesurer objectivement l'erreur réelle du couplage entre ICP et Kalman, il est donc difficile d'évaluer le gain obtenu grâce à notre version d'ICP.

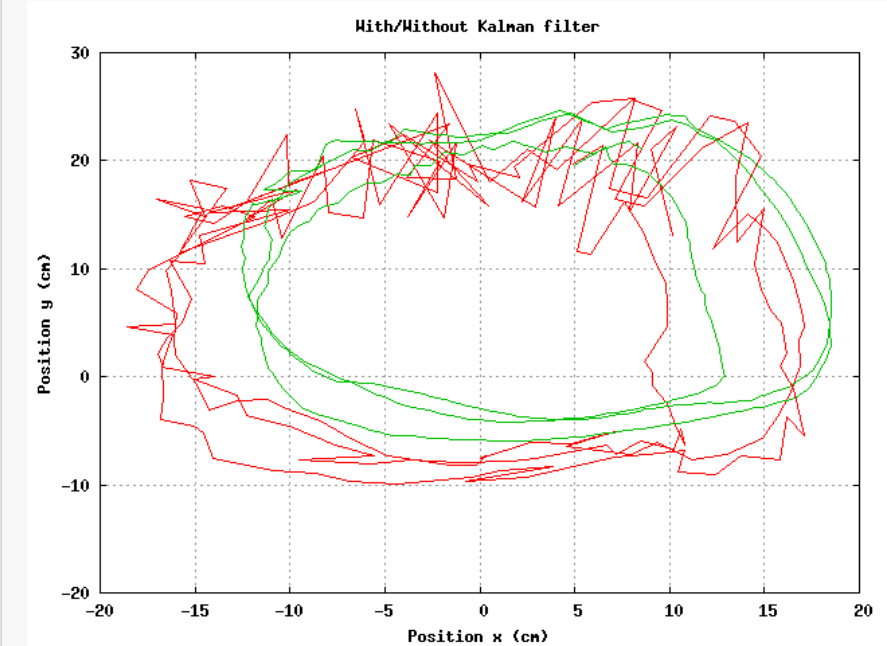


Fig. 5.6: Superposition de la trajectoire initiale, et de la trajectoire corrigée par le filtre de Kalman combiné à notre version d'ICP.

5.3.2 Reconnaissance de postures

Le suivi des mouvements des mains est essentiel pour la reconnaissance de postures et de gestes. Nous venons de voir que le couplage du filtre de Kalman et de l'algorithme ICP permet d'obtenir des résultats satisfaisants lorsque le

matériel est sensible au bruit. Dès lors que cette première étape est validée, nous pouvons nous intéresser à la reconnaissance elle-même.

La reconnaissance de postures est une problématique consistant à pouvoir déterminer, à un instant T, la configuration morphologique des phalanges d'une main, en l'associant à une posture théorique prédéfinie. Cette problématique peut être scindée en deux parties distinctes. D'une part, le calibrage des gants de données sert à prendre en considération la morphologie de chaque utilisateur. D'autre part, des méthodes de reconnaissance de postures permettent d'associer une commande à une configuration spatiale d'une main.

Calibrage des angles de flexion

Les angles de flexion sur différents gants de données varient selon :

- le type de capteurs utilisés ;
- la morphologie palmaire ;
- le nombre de capteurs utilisés.

Ce type de variation limite la reconnaissance de position ; un individu ayant une grande main pourra faire varier les valeurs de capteurs sur leur amplitude complète tandis qu'un autre sera limité sur une partie de celle-ci. Le nombre de capteurs influe également sur la précision des informations. Cependant, il n'est pas envisagé de développer une méthode pour un type de gant particulier : notre calibrage doit être adaptable quel que soit le matériel utilisé. Il s'agit donc ici d'adapter les données enregistrées par les gants de données – tout type de capteurs confondus – à la morphologie de tout utilisateur.

Plusieurs types de calibrage existent :

- Le calibrage durant l'utilisation : l'application s'adapte progressivement aux données qu'elle reçoit.
- Le calibrage préalable : l'utilisateur lance préalablement un programme qui détermine l'échelle de variation des valeurs de chaque capteur, puis il peut lancer l'application souhaitée.

Nous utilisons un programme de calibrage avant le lancement de l'application, il n'y aura donc aucune adaptation en cours d'exécution.

Dans le domaine du calibrage, différentes méthodes ont été étudiées ; on citera par exemple une approche visuelle avec régression linéaire [CD00], un apprentissage basé sur un réseau de neurones utilisant une main mécanique [FVdSH98], et dans le cas où les gants possèdent des capteurs d'abduction, on pourra consulter la méthode de Kahlesz [KZK04]. Nous nous basons uniquement sur les valeurs retournées par le matériel, puisque dans notre cas, ce dernier ne possède aucun capteur d'abduction.

Notre outil est assez simple : nous récupérons les valeurs de flexion renvoyées par chaque capteur et enregistrons les minimums et maximums de chacun d'eux. L'utilisateur doit bouger sa main pendant une durée d'une trentaine de secondes

en alternant des positions "poing ouvert" et "poing fermé", ainsi que d'autres positions permettant de stimuler le plus possible chaque capteur. Notre objectif est de déterminer les angles minima et maxima, pour chaque angle de flexion de chaque phalange, afin de mesurer correctement les valeurs angulaires des doigts lors de l'exécution du programme. Le calibrage ainsi réalisé facilite la reconnaissance de postures et de gestes.

Ainsi, à l'aide d'un fichier de paramètres – propres à chaque utilisateur, l'application sera en mesure d'adapter l'échelle de variation des capteurs à la morphologie de la main. Précisons que notre méthode de calibrage comporte un risque de mauvaise lecture du minimum et du maximum des angles lorsqu'un utilisateur n'enfile pas toujours un gant de la même manière. La figure 5.7 montre le fonctionnement de notre outil de calibrage. On peut voir que les valeurs minimales et maximales sont étendues pour couvrir la nouvelle plage des valeurs de flexion possibles.

Afin de valider notre méthode nous effectuons divers tests qui consistent à vérifier la stabilité de notre méthode selon différentes morphologies dans le temps. Le protocole suivi est le suivant :

- Le but de l'application est expliqué à l'utilisateur et l'emplacement théorique des capteurs de flexions est indiqué sur un un avatar virtuel de la main.
- Chaque utilisateur procède à une série de mesures successives ; entre chacune d'elles il lui est demandé de retirer puis de remettre le gant, afin de s'assurer que le calibrage est stable dans le temps. On mesure l'écart relatif entre les différentes mesures pour s'assurer de la validité de notre méthode.

Nous avons effectué des mesures sur 5 utilisateurs, tous droitiers, et dont la taille des mains varie entre 17,5 cm et 22 cm. Nous avons utilisé deux gants de données : le gant P5 et le X-ist. Le premier est une structure semi-rigide plastifiée que l'utilisateur peut régler en fonction de sa morphologie. Le second est en tissu et s'enfile sans réglage préalable.

Pour chacun des 5 utilisateurs, nous procédons à 5 séries de 6 mesures successives – une mesure pour le poing fermé, puis une mesure pour la main à plat, etc.. Le tableau suivant montre la synthèse des résultats que nous avons obtenus. Il indique pour chaque utilisateur, selon la taille de sa main, pour les deux gants, les variations moyennes en degrés enregistrées par rapport aux angles min. et max. calibrés initialement. Les valeurs min. correspondent aux angles de la main à plat, et les valeurs max. aux angles du poing fermé.

Utilisateur	1	2	3	4	5
Taille main	17,5 cm	18,2 cm	21,2 cm	19,4 cm	22 cm
P5 - min	5,51	4,7	10,61	6,02	12,33
P5 - max	3,67	3,1	1,5	2,07	2,01
X-ist - min	3,51	3,11	7	5,74	8,08
X-ist - max	3,44	3,07	8,49	4,03	10,2

Nous pouvons noter que le P5 semble mieux convenir aux personnes ayant des mains dont la taille tourne aux alentours de 18 cm. Pour ceux ayant une

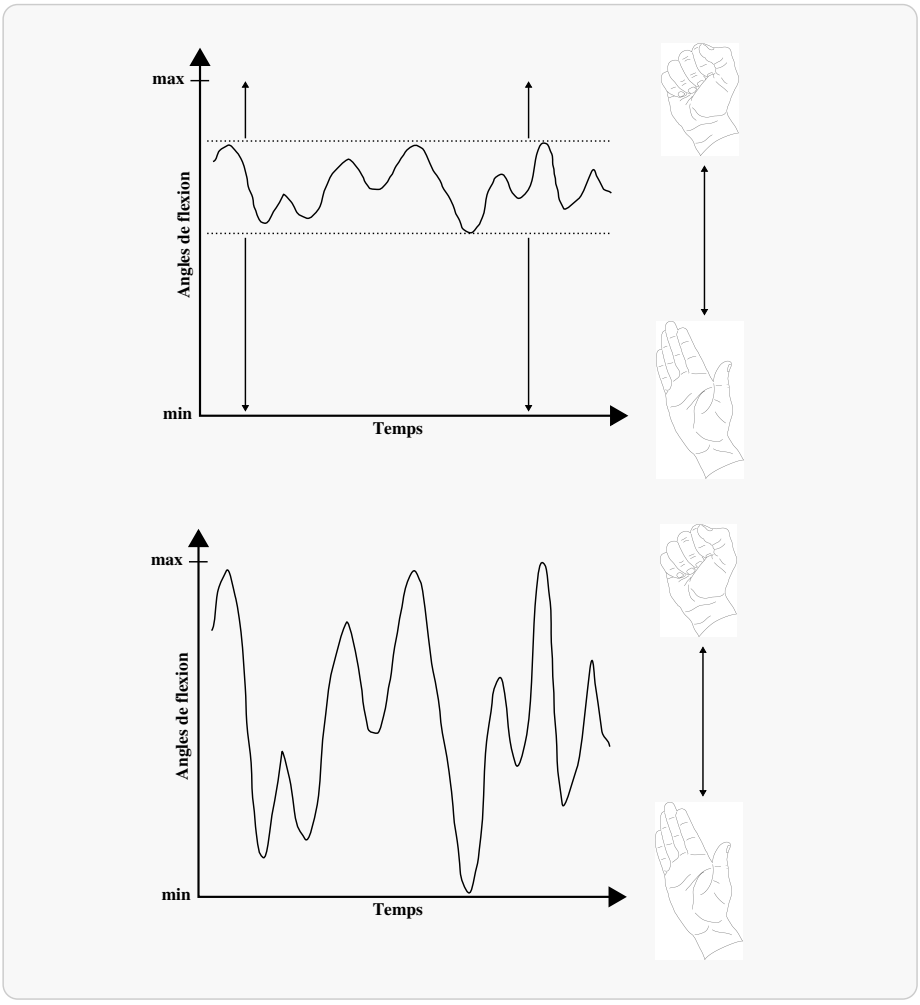


Fig. 5.7: Calibrage des gants de données (image du haut : avant le calibrage, image du bas : après le calibrage).

grande main, les variations sont les plus importantes. Notons que pour ce gant, le gant sature lorsque le poing est fermé ; il est donc logique que, pour les valeurs maximales, les écarts constatés soient faibles. Le gant X-ist semble également affecté par ce phénomène, et l'utilisateur ayant la plus grande main a d'ailleurs noté qu'il avait quelques difficultés à enfiler convenablement ce gant. Les résultats montrent que la variation moyenne au niveau de l'ensemble des angles de flexion est assez faible, tant que l'utilisateur n'a pas une main trop grande – c'est-à-dire, pour les deux gants, plus de 21 cm. Notre méthode de calibrage semble donc suffisamment stable et efficace pour la reconnaissance des gestes et des postures.

Reconnaissance de postures

Définition Une **posture** est l'élaboration et le maintien actif de la configuration des différentes phalanges d'une main dans l'espace.

Une posture correspond à une configuration fixe des phalanges de la main dans le temps et l'espace, comme par exemple fermer le poing. Sur cet exemple, même si l'utilisateur bouge sa main dans l'espace, les angles de flexion ne se modifient pas ; il s'agit alors d'une posture.

Nous nous intéressons à la reconnaissance des postures à l'aide de gants de données ; il s'agit à un instant T de pouvoir déterminer la signification de la configuration des phalanges de la main. Dans notre cadre expérimental, nous avons retenu 9 postures simples que l'utilisateur peut accomplir et employer dans son application, et dont voici la liste :

- poing fermé ;
- pointer avec l'index ;
- pointer avec le majeur ;
- pointer avec l'index et le majeur ;
- pointer avec le petit doigt ;
- la main à plat ;
- pointer avec le pouce ;
- plier uniquement le pouce ;
- pointer avec le pouce et le petit doigt.

Le choix de ces 9 postures distinctes a été réalisé de manière à ce qu'elles soient bien démarquées les unes des autres, afin de faciliter la reconnaissance, et minimiser les recoupements. Nous proposons deux méthodes de reconnaissances, que nous allons comparer :

- La méthode brute : simple récupération de valeurs et test d'identification.
- L'utilisation d'un réseau de neurones : comparaison avec des valeurs idéales apprises par le réseau de neurones.

Il existe de nombreuses techniques de reconnaissance dans la littérature. Pour une reconnaissance de posture effectuée sur la logique floue et donnant des résultats efficaces sur des données bruitées, on pourra consulter la méthode développée par Tsang [TS98]. Nous avons retenu les deux précédentes pour des raisons de coût de développement.

Reconnaissance de postures par méthode brute La méthode de reconnaissance par méthode brute est basée sur un apprentissage durant lequel l'utilisateur doit répéter plusieurs fois des postures identiques afin que l'application apprenne à les reconnaître. Durant ces répétitions, un enregistrement des angles minimaux et maximaux est effectué sur chaque phalange. Lors de la phase d'utilisation, elle cherche alors à comparer les valeurs d'angles de l'utilisateur à celles préalablement enregistrées. Si les valeurs d'angles sont comprises dans les bornes résultantes de l'apprentissage, alors la posture est reconnue.

Nous avons effectué des tests sur un groupe de 7 personnes, toutes droitières. La démarche de test se déroule en deux étapes. La première étape consiste en une phase d'apprentissage, chaque individu répète trois fois chacune des postures retenues et maintient la position durant 5 secondes en portant un gant de données ; un fichier de configuration personnalisé est alors créé et peut être chargé pour des futures utilisations. La deuxième étape consiste en un test d'efficacité de cette méthode. L'utilisateur est invité à réitérer l'opération précédente ; cette fois-ci l'application se base sur le fichier de configuration créé précédemment pour tenter de reconnaître la posture effectuée. Le tableau 5.3.2 présente les taux de réussite de reconnaissance par méthode brute.

Posture :	Fist	Index	Middle	Two	Little	Flat
Reconnaissance (%) :	100	66.67	47.61	33.33	57.14	76.19

Posture :	Thumb	Not Thumb	Thumb and Little
Reconnaissance (%) :	85.71	66.67	85.71

Tab. 5.2: Taux de reconnaissance des postures par méthode brute.

On constate que les taux de reconnaissance *via* la méthode brute sont assez moyens – 68.78% de reconnaissance en moyenne, hormis pour le poing. Ces valeurs s'expliquent par le fonctionnement même de la méthode ; en effet, lors du calibrage, l'utilisateur réalise plusieurs fois le geste, ce qui détermine une plage d'angles minimaux et maximaux. Or, la sensibilité est extrême, puisqu'une posture approximative ne sera tout simplement pas reconnue, même si elle se rapproche fortement de la posture souhaitée et calibrée.

Reconnaissance de postures avec un réseau de neurones Nous venons de présenter une première méthode de reconnaissance de postures dite brute. Cette façon de procéder ne permet de reconnaître convenablement plus de 2 postures différentes : la main ouverte et la main fermée. Les autres postures ont tendance à se recouper, notamment du fait que les capteurs qui enregistrent les angles de flexion des phalanges ne sont pas très précis. Nous souhaitons utiliser un réseau de neurones pour améliorer la reconnaissance des postures. D'autres auteurs ont réalisé des travaux similaires [SW00, SSK01].

Le principe de notre réseau de neurones est assez simple. En entrée, nous introduisons l'ensemble des valeurs d'angles de notre gant de données. Cet ensemble de neurones passe par une couche cachée de n neurones afin d'activer l'un des 9 neurones de la couche de sortie, chacun d'entre eux correspondant à

une posture. L'apprentissage est effectué de manière totalement indépendante de l'utilisateur ; des valeurs d'angles idéales pour chaque posture sont données en entrée tout en indiquant quel neurone de sortie doit être activé. Tant que la propagation des informations d'entrées n'active pas le bon neurone de sortie avec une certaine précision, les poids de chaque liaison entre les neurones sont modifiés.

Pour notre protocole expérimental nous avons pris les valeurs suivantes : 100 neurones sur la couche cachée et l'apprentissage s'effectue jusqu'à atteindre une erreur globale inférieure à 0.5%. Nous avons également essayé plus ou moins de neurones en couche cachée, mais les résultats ne se sont pas révélés meilleurs. Une posture n'est considérée comme reconnue que si le réseau de neurones donne un degré de certitude supérieur à 80%. Le programme de test de reconnaissance est le même que celui utilisé pour la méthode brute. Le tableau 5.3.2 présente les taux de réussite de reconnaissance par méthode neuronale.

Posture :	Fist	Index	Middle	Two	Little	Flat
Reconnaissance (%) :	100	100	66.67	80.97	85.71	76.19

Posture :	Thumb	Not Thumb	Thumb and Little
Reconnaissance (%) :	100	85.71	85.71

Tab. 5.3: Taux de reconnaissance des postures par méthode neuronale.

Les résultats obtenus par la méthode du réseau de neurones est plus efficace que la méthode brute, puisque le taux de reconnaissance moyen est de 86.77% sur nos 10 utilisateurs. Ce résultat est nettement supérieur aux 68.78% de la méthode brute présentée précédemment. Pour les deux méthodes, la majorité des cas de postures non reconnues sont dus soit à des problèmes de capteurs de flexion – le matériel n'est pas totalement fiable, soit à la morphologie inadaptée de certains utilisateurs aux gants de données. Dans ce cas, le calibrage montre des limites qui sont relatives au matériel.

Pour améliorer la reconnaissance, nous proposons d'établir un profil préalable, propre à chaque utilisateur. Les valeurs ainsi obtenues sont ajoutées aux valeurs idéales pour l'apprentissage du réseau de neurones. Ce mécanisme permet essentiellement de corriger des erreurs liées à une morphologie particulière qui donnent des angles de flexions trop éloignés par rapport aux angles idéaux.

5.3.3 Reconnaissance de gestes

Définition Un geste est une succession de mouvements à laquelle est associée une signification particulière, dans le but d'accomplir une tâche.

La reconnaissance de postures permet de détecter des configurations statiques de la main dans l'espace. Celles-ci permettent donc de lancer des commandes au sein de l'application. L'introduction des gestes sert à étendre les possibilités d'interaction en proposant ce que l'on appelle un langage gestuel opératif. Un geste est inscrit dans un langage défini par une grammaire ; un

geste est donc un mot du langage, et nous souhaitons mettre en œuvre un système de reconnaissance propre à détecter efficacement l'intention de l'utilisateur, avant d'interpréter la nature de ses actions.

La reconnaissance de gestes que nous souhaitons mettre en place se base sur un système de grammaire prenant en compte les postures, les orientations et les positions que prend la main au cours du temps. Un réseau de neurones tel que celui utilisé pour les postures ne serait pas cohérent ici car nous n'avons pas de "gestes exemple" sur lesquels un apprentissage est possible, comme dans le cas de la langue des signes [MT91]. En effet, un geste dans notre situation pourra prendre des paramètres différents tout en conservant la même signification. On peut parler alors d'équivalence entre différents gestes.

Notre grammaire repose sur la séparation des différentes composantes du mouvement. Dans l'implantation actuelle, les informations concernant le déplacement et la posture de la main sont considérées de manière totalement distincte. Le geste correspond au regroupement de ces informations successives en une seule, à l'aide d'une grammaire qui permet de simplifier et manipuler chaque information (orientation de la main et posture courante, etc.). Nous proposons une notation iconique simple, qui se veut accessible rapidement à des utilisateurs non spécialistes : ceux-ci pourront définir leurs propres gestes à l'aide d'une grammaire élémentaire.

Déplacements de la main Plutôt que de considérer l'orientation du poignet comme des valeurs variant de manière continue, nous définissons un ensemble de 26 translations possibles, ainsi que 3 axes de rotation de la main sur elle-même. Nous choisissons de nommer cette représentation la boussole $3D$; celle-ci nous permet donc de décrire un déplacement comme étant :

- une direction de translation selon 26 directions `N` – North, `S` – South, `E` – East, `W` – West, `H` – Heaven, `G` – Ground, `NE`, `NW`, `NH`, `NG`, `NEH`, `NEG`, `NWH`, `NWG`, `SE`, `SW`, `SH`, `SG`, `SEH`, `SEG`, `SWH`, `SWG`, `HE`, `HW`, `GE`, `GW` ;
- une rotation selon 3 axes : $\{x, y, z\}$ – yaw, pitch, roll.

Pour définir un geste particulier, on considère une suite d'informations de la boussole $3D$, et de postures que le système est capable de reconnaître.

Logique de la grammaire Pour la reconnaissance des gestes, la détection de début et de fin de geste pose de sérieux problèmes. En effet, comment déceler la posture qui correspond au début du geste sans la confondre avec une simple configuration de la main de l'utilisateur issue du flot des mouvements naturels du corps ? Bien entendu, nous pourrions définir une posture particulière qui indique le début et la fin du geste ; cependant, nous aimerions disposer de toutes les postures possibles et ne pas demander à l'utilisateur de signifier clairement le début et la fin d'un geste. Le reconnaissance des gestes s'inscrit donc dans le flot continu des mouvements des mains, et notre système doit être en mesure d'éviter de repérer de faux débuts de gestes.

Pour cela, nous définissons un automate à états. Le principe de notre algorithme est de supposer que ce qui semble être un début de geste n'en est pas nécessairement un. Il faut, à chaque identification de geste, réinitialiser l'état de l'automate afin de pouvoir gérer d'autres gestes. À chaque état de l'automate sont associées les modifications effectuées depuis l'état précédent. Cette association nous oblige à avoir une classe d'équivalence assez souple entre états mais sans pour autant supprimer des informations comme cela est illustré sur la figure 5.8.

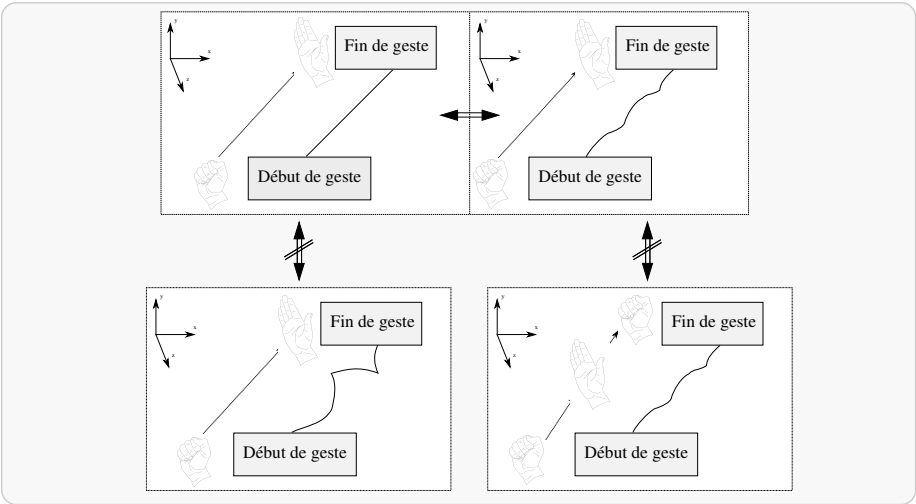


Fig. 5.8: Équivalence des gestes : le déplacement rectiligne équivaut au déplacement rectiligne bruité, mais est différent du déplacement en S.

Un état est donc défini par :

- une des directions indiquées par la boussole $3D$;
- un changement de rotation supérieur à un angle donné ;
- une posture.

Par exemple, un geste de translation associé à une ouverture de la main – comme lancer un objet sur le sol – devra correspondre aux états suivants :

1. main fermée translatée dans la direction `NG` sur une distance k ;
2. main fermée translatée dans la direction `G` sur une distance l ;
3. main ouverte translatée dans la direction `G` sur une distance m ;
4. début d'un autre geste.

Construction de l'automate Il est impératif de contraindre la grammaire de manière à ce qu'elle soit déterministe. Il ne doit pas exister d'état qui aboutisse à un puits, et duquel on ne puisse plus sortir. De plus, il faut aussi s'assurer qu'un geste ne se retrouve pas inclus dans un autre, auquel cas on ne pourrait

jamais effectuer le geste le plus long.

La problématique la plus importante dans la reconnaissance de gestes est de réussir à cibler le commencement et la terminaison d'un geste de manière naturelle. Notre automate a pour principe d'utiliser des jetons d'historique qui permettront le commencement d'un geste puis son interruption par l'exécution d'un autre. Le fonctionnement des jetons d'historique est décrit de la façon suivante :

- l'automate est dans un état initial ;
- un changement de posture/mouvement est détecté ;
- s'il y a un état de l'automate qui permet la reconnaissance de ce déplacement, on place alors le jeton A dans l'état correspondant ;
- un nouveau changement de posture/mouvement est détecté ;
- le jeton A est déplacé en conséquence dans un nouvel état ;
- dans le même temps on aura initialisé un jeton B dans l'état initial de l'automate que l'on déplacera à partir du deuxième déplacement repéré ;
- on réitère cette création de jeton si aucun jeton ne se trouve dans l'état initial ;
- si un jeton arrive dans un état final : un geste a été identifié ; on effectue l'action correspondante et l'on réinitialise notre automate.

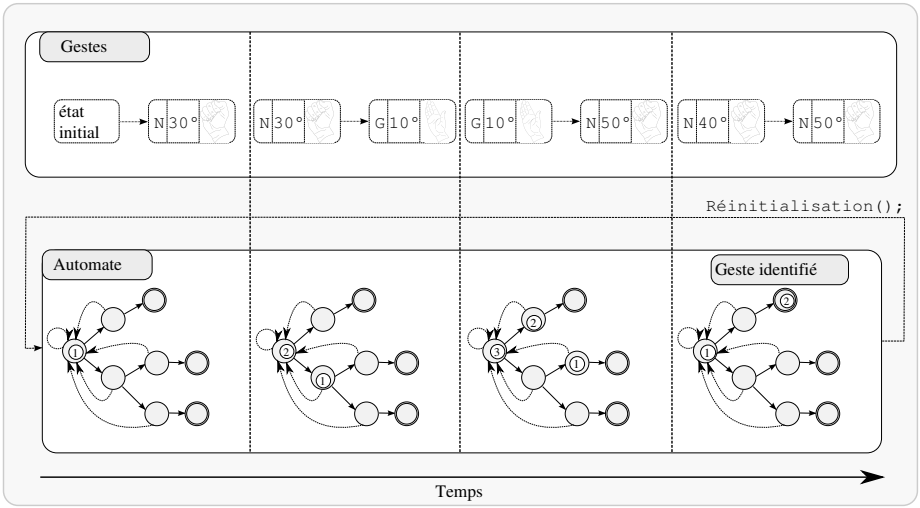


Fig. 5.9: Méthode de reconnaissance de gestes par automate à états.

Nous n'avons pas encore implanté cet algorithme de reconnaissance de gestes au sein de la vrLib. Nous aimerions avant tout nous concentrer sur l'amélioration de le reconnaissance de postures notamment à l'aide d'une méthode statistique de type chaîne de Markov cachée. En effet, cette méthode semble plus efficace que le réseau de neurones dont nous disposons actuellement, mais reste assez lourd à mettre en œuvre.

5.3.4 Conclusion

Nous nous intéressons à la reconnaissance de postures et de gestes à l'aide de gants de données qui retournent des valeurs d'angles de flexion des mains de l'utilisateur. Nous avons expliqué comment le couplage des algorithmes ICP et Kalman permet de stabiliser les mouvements des mains dans l'espace et le temps. Cette association donne des résultats très encourageants, notamment lorsque le matériel de suivi des mouvements est peu précis.

Nous avons présenté notre méthode de calibrage des gants, qui se veut à la fois simple et efficace. Nous avons montré qu'elle atteint les résultats escomptés, dès lors que la morphologie de l'utilisateur s'adapte bien à la structure physique du gant. Lorsque ce n'est pas le cas, le calibrage ne peut rattraper les erreurs de mauvais placement des capteurs de flexion sur les phalanges de la main.

La reconnaissance des postures est très intéressante, et a été rapidement intégrée au sein de la vrLib de façon à proposer une interaction gestuelle simple et efficace. Le programmeur peut choisir entre deux méthodes. La première est dite brute, parce qu'elle compare les angles de flexion courants des doigts à des valeurs pré-enregistrées pour chaque utilisateur. Cette technique est rapide, et peu gourmande en ressources, mais elle ne se prête qu'à des cas de reconnaissance simple avec peu de postures ; essentiellement la main ouverte, et la main fermée. Des problèmes de recoupement peuvent apparaître lorsque des plages de valeurs se superposent, ce qui peut arriver lorsque les postures choisies sont trop proches.

La seconde méthode est basée sur l'utilisation d'un réseau de neurones. Les taux de reconnaissance sont bons, et il n'y a pas de problème de recoupement des postures reconnues contrairement à la technique brute. Il est dès lors possible de reconnaître plus de postures différentes – nous en avons retenu 9, mais la précision du matériel et les différences morphologiques parfois importantes entre les utilisateurs peuvent faire échouer cette méthode. Nous souhaitons ajouter, dans un avenir proche, une troisième méthode statistique basée sur les chaînes de Markov cachée. Celle-ci pourrait améliorer le taux de reconnaissance des postures considérées.

Le dernier sujet que nous avons développé ici est la reconnaissance de gestes. Nous avons présenté la méthode de la boussole $3D$ couplée à un automate à états, parcouru à l'aide de jetons d'historique. Une grammaire simplifiée permet de décrire rapidement des gestes dans un fichier de configuration. Pour le moment cette méthode n'a pas été implantée dans notre librairie ; nous espérons dans un premier temps améliorer la reconnaissance de postures. Le point fort de cette technique est de ne pas avoir à expliciter le début et la fin du geste.

5.4 Rayon déformable

Nous avons présenté notre travail sur la reconnaissance de postures et de gestes ; celui-ci sert de base à la conception de certaines techniques d'interaction comme le rayon déformable qui fait l'objet de cette section. Cette métaphore

repose en effet sur l'exploitation conjointe des deux mains. Il s'agit d'une extension à la technique du lancer de rayon classique, et cette approche constitue une technique d'interaction innovante pour la navigation et la sélection en environnement virtuel. Elle combine les bénéfices suivants :

- Elle permet une sélection simple d'objets semi-cachés par d'autres objets en contournant les obstacles.
- Elle est simple et multi-sélection.
- Il s'agit technique de lancer de rayon précise, que ce soit sur les objets proches ou lointains.
- Elle fournit une technique de navigation contrainte le long d'une courbe.
- Elle sert à planifier la navigation.

Nous allons commencer par analyser comment le lancer de rayon déformable se place par rapport aux techniques de sélection et de navigation existantes. Nous décrivons ensuite toutes les capacités de notre technique du lancer de rayon déformable, avant de conclure sur certaines perspectives intéressantes.

5.4.1 Travaux précédents

Aujourd'hui, la disponibilité d'environnements graphiques de haute résolution, comme le *Workbench* ou le CAVE, permet de créer des environnements virtuels dans lesquels un utilisateur doit se déplacer et sélectionner, puis manipuler des objets. La navigation est un processus qui modifie son propre point de vue, et la sélection permet d'indiquer à l'application quels sont les objet virtuels que l'on souhaite manipuler. Bien entendu, un bon point de vue est un préalable nécessaire à une manipulation efficace. Comme dans la vie de tous les jours, la manipulation se fait par étapes successives : on se déplace, on manipule, on repose l'objet, on se re-déplace, puis on manipule à nouveau. Il est nécessaire de réduire le nombre d'étapes, afin que l'interaction soit la plus aisée possible.

De manière générale, sur la plupart des configurations de réalité virtuelle, seules deux ou trois techniques sont utilisées en pratique pour sélectionner et manipuler un ou plusieurs objets ; soit à l'aide de la technique de la main virtuelle, soit *via* la technique du lancer de rayon. Bolt [Bol80] introduit le pointeur virtuel en tant que métaphore de sélection, dans laquelle un rayon virtuel part de la main de l'utilisateur pour attraper les objets de la scène. Plusieurs auteurs [MCR90, ZBM94, Min95] ont étendu cette méthode tout en conservant sa simplicité : un rayon rectiligne qui traverse la scène virtuelle.

5.4.2 Description

Ce type de métaphore engendre un certain nombre de problèmes inhérents à leur conception. Le premier est que plus un objet est loin et/ou petit, plus la sélection est délicate, puisque les mouvements de l'utilisateur s'amplifient sur la longueur du rayon. Le second problème survient lorsque l'objet d'intérêt est caché derrière un autre objet ; l'utilisateur peut distinguer les deux objets, mais le manque de stabilité et sa position dans l'environnement l'obligent à se déplacer pour se rapprocher et contourner l'obstacle, avant que la sélection ne débute. Olwal *et al.* [OF03] proposent de résoudre partiellement ce problème en courbant

un rayon virtuel de longueur fixe décrit à l'aide d'une spline. Cependant, cette technique montre ses limites lorsqu'il s'agit de contourner plusieurs obstacles à la suite, et la longueur du rayon, limitée, influe directement sur sa courbure.

La technique du lancer de rayon déformable permet de sélectionner des objets cachés par d'autres, en décrivant une courbe de forme quelconque dans la scène virtuelle. Sa longueur varie dans le temps en fonction de la position des deux mains de l'utilisateur dans l'espace. Le rayon vient littéralement embrocher un ou plusieurs objets qu'il rencontre sur son passage, même si ceux-ci sont loin les uns des autres. La taille du rayon peut soit augmenter, soit diminuer, et dans ce dernier cas, il est possible d'obtenir un équivalent à l'opération d'annulation de la sélection.

La navigation est un prérequis essentiel à toute sélection lorsque l'utilisateur est mal placé dans l'environnement virtuel par rapport à l'objet d'intérêt. Notre technique du lancer de rayon déformable peut servir à explorer un lieu de la scène, à distance en manipulant le bout du rayon, comme si l'on déplaçait une caméra. Par exemple, dans une opération chirurgicale, le praticien souhaite explorer une reproduction virtuelle du réseau sanguin d'un patient pour planifier une intervention. Notre approche permet de tracer un chemin d'exploration de forme quelconque, puis de l'enregistrer pour le réutiliser ensuite.

5.4.3 Fonctionnement

Le rayon déformable est composé de morceaux de courbe qui sont additionnés les uns aux autres dans la scène $3D$, comme le montre schématiquement la figure 5.10. La manipulation du rayon est réalisée à l'aide des mains de l'utilisateur en même temps. Nous pensons, de la même manière que Mine *et al.* [MBJS97], que l'utilisation conjointe des deux mains est une méthode naturelle et intuitive pour l'utilisateur. La position des mains définit un vecteur qui est utilisé pour modifier le bout du rayon, c'est à dire les points de contrôle du morceau de spline définissant le bout durayon. La direction du dernier morceau du rayon est donnée par la direction du vecteur. La vitesse d'agrandissement et de réduction du rayon est proportionnelle à la distance entre les deux mains.

Notre approche est fonctionnelle tant pour les gauchers que pour les droitiers. L'utilisateur débute l'interaction en plaçant ses mains devant lui, la main non dominante – droite pour un gaucher, gauche pour un droitier – définit la base du vecteur, et la main dominante donne la direction et la longueur du vecteur. Pour initier et terminer l'utilisation de cette métaphore, l'utilisateur doit fermer le poing de la main droite, hormis son index et son majeur, comme l'illustre la figure 5.11.

Trois modes d'utilisation sont disponibles :

- En conservant les deux mains en contact, la taille du vecteur est nulle et le rayon n'est pas modifié.
- En fermant le poing de la main non dominante, le rayon se réduit en taille.
- En ouvrant la main non dominante, le rayon s'allonge dans la scène virtuelle.

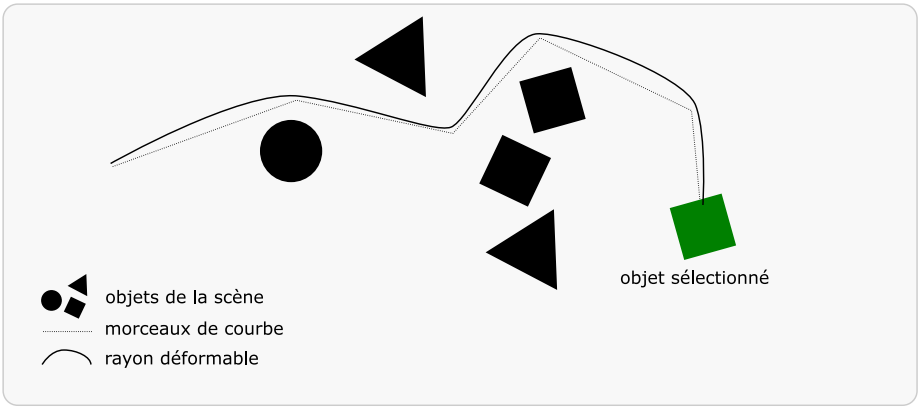


Fig. 5.10: Métaphore du lancer de rayon déformable.

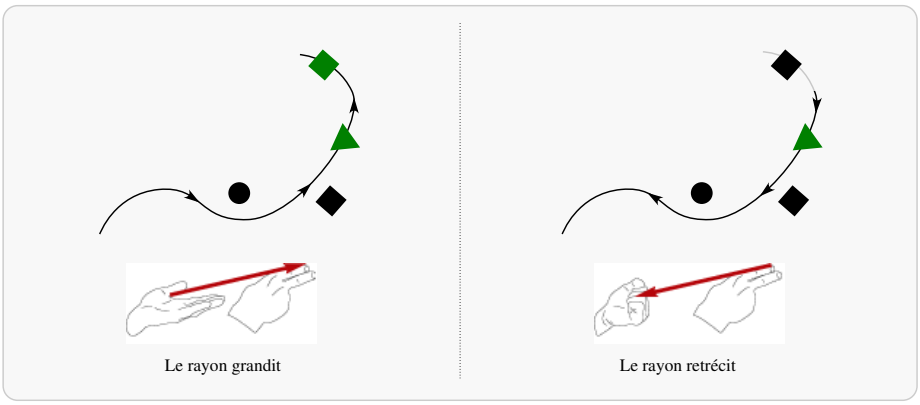


Fig. 5.11: Définition du vecteur de navigation à l'aide des deux mains.

Regardons plus attentivement le fonctionnement de l'utilisation du rayon pour réaliser une tâche de sélection et une tâche de navigation.

Sélection

La technique traditionnelle pour sélectionner un objet est basée sur un rayon rectiligne dont la longueur est fixe. Lorsqu'un objet d'intérêt est caché par un autre objet, l'utilisateur doit, dans un premier temps, changer son point de vue avant d'être capable de réaliser la sélection. Olwal *et al.* [OF03] proposent une première approche en imaginant un rayon courbe ou flexible défini à l'aide d'une Spline quadratique. Le premier problème de cette méthode est qu'il est impossible de courber le rayon plus d'une fois, dans une situation qui nécessite plusieurs points d'inflexion. Le second problème est que la longueur du rayon est fixe. Notre approche est de considérer plusieurs courbes Spline qui sont mises bout à bout en ajoutant ou en enlevant des points de contrôle pour modifier la taille du rayon.

En faisant passer le rayon par un ou plusieurs objets, l'utilisateur est capable de sélectionner aussi bien un seul que plusieurs objets différents. Il est également possible de désélectionner le dernier objet enregistré en raccourcissant la courbe dans l'espace.

La métaphore du lancer de rayon déformable permet à l'utilisateur de rester précis durant la tâche de sélection, même sur les objets distants, car il contrôle le dernier segment de courbe. Il n'y a aucune amplification des mouvements naturels de postures – légers tremblements de la main, comme c'est le cas avec le lancer de rayon classique. Notre approche réduit donc le besoin de se déplacer dans la scène avant la tâche de sélection, lorsque l'objet est partiellement caché par un autre.

Navigation

La technique du lancer de rayon déformable permet également d'explorer un environnement virtuel. Dans sa version navigation, cette technique dispose d'une caméra placée au bout du rayon. Elle offre à l'utilisateur un second point de vue qui est affiché dans un coin de la scène. La métaphore devient alors un outil d'exploration qui enregistre son propre chemin de navigation.

Ainsi, notre technique d'interaction autorise l'exploration d'une scène $3D$ *via* un second point de vue. On peut la comparer au procédé cinématographique de *travelling*, où la caméra est attachée sur un rail et sert à suivre le mieux possible les déplacements des acteurs. Un chirurgien pourra par exemple bénéficier du lancer de rayon déformable pour l'exploration d'un vaisseau sanguin dans le but de planifier une opération.

5.4.4 Conclusion

Nous venons de décrire la métaphore du lancer de rayon déformable, qui est employée pour les tâches de sélection et de navigation. Dans le cas de la

sélection, c'est un outil très intéressant qui permet d'attraper un ou plusieurs objets, que ceux-ci soient partiellement cachés ou non, et quelle que soit leur distance avec l'utilisateur. La technique que nous proposons peut également servir à la tâche de navigation planifiée, et à l'exploration, le long d'une courbe $3D$ de forme quelconque. Enfin, nous pouvons imaginer une version dédiée à la manipulation contrainte ; dans ce cas, l'objet est sélectionné pour se déplacer le long de la courbe dessinée par l'utilisateur.

5.5 Volumes de sélection

Dans un environnement virtuel, lorsque plusieurs objets doivent être sélectionnés simultanément, il peut être intéressant de disposer d'un volume de sélection. Nous proposons deux techniques de sélection multi-objets par volume ; la première est la boîte englobante et la seconde, la métaphore du lasso $3D$.

5.5.1 Par boîte englobante

L'idée de sélectionner plusieurs objets en même temps est une fonctionnalité intéressante lorsque l'on travaille sur les objets d'une scène virtuelle. Dans les interfaces $2D$, la sélection multi-objets passe par la définition d'une zone rectangulaire à l'aide de la souris : tous les objets qui tombent dans cette région sont sélectionnés. Nous proposons d'étendre cette méthode à la $3D$ en définissant un volume de sélection parallélépipédique à l'aide des deux mains de l'utilisateur.

Le manipulateur fixe deux points dans l'espace, qui correspondent à deux sommets opposés du parallélépipède. Tous les objets qui se situent à l'intérieur du volume sont alors sélectionnés. La figure 5.12 montre le principe de fonctionnement de cette technique d'interaction. Le volume est affiché en pointillé tant que l'utilisateur le définit, puis les arêtes deviennent pleines pendant 2 secondes lorsqu'il cesse de manipuler le parallélépipède.

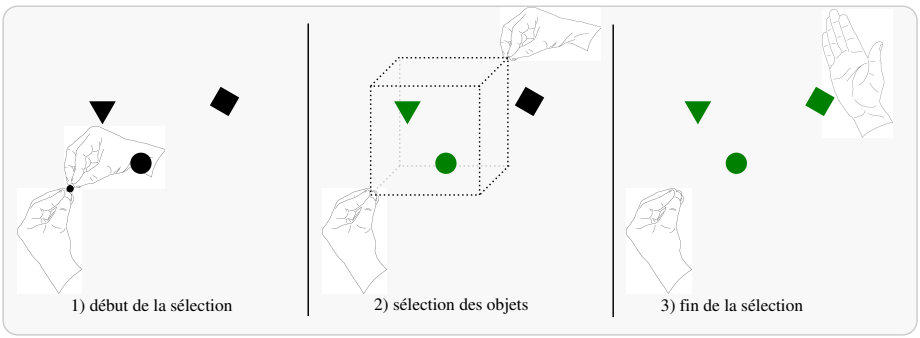


Fig. 5.12: Métaphore de sélection par boîte englobante.

En pratique, nous utilisons la technique de la boîte englobante pour faciliter la sélection d'un ensemble de sommets sur un modèle géométrique. Initialement, la technique du lancer de rayon était longue et fastidieuse, car l'utilisateur devait viser très précisément chacun des sommets du maillage. L'approche par

volume de sélection parallélépipédique permet de réduire sensiblement le temps de sélection lorsque les sommets sont regroupés, au détriment de la précision.

5.5.2 Par lasso $3D$

La technique de sélection par boîte englobante est simple à mettre en œuvre. Cependant, elle ne permet de définir qu'un parallélépipède, et il peut s'avérer utile d'avoir un volume de sélection de forme différente et surtout orientable dans l'espace. Nous proposons d'étendre l'outil lasso utilisé dans les logiciels $2D$ pour sélectionner une zone d'apparence arbitraire. Cette technique permet d'entourer les objets d'intérêt à l'aide d'un simple trait. Pour initier le lasso, l'utilisateur appuie sur un bouton de la souris et le maintient enfoncé. Lorsqu'il relâche le bouton, la ligne dessinée se forme par un trait entre le début et la fin du lasso. La figure 5.13 illustre le fonctionnement du lasso $2D$.

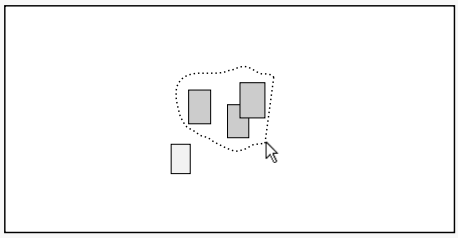


Fig. 5.13: Technique de sélection mutliple par lasso $2D$.

En $3D$, l'utilisation conjointe des deux mains permet de définir un volume de sélection. Pour éviter de rendre la métaphore trop complexe, nous avons choisi de restreindre la forme à une ellipsoïde, et son équation est de la forme :
$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$
où a, b et c sont des paramètres strictement positifs donnés, égaux aux longueurs des demi-axes de l'objet.

À l'initialisation du lasso $3D$ les doigts de chaque main sont regroupés autour du pouce, et les deux pouces se touchent, ce qui a pour effet de définir le centre de l'ellipsoïde. L'éloignement de chaque main est pris en compte pour déterminer les valeurs de x, y et z. Pour terminer la définition du volume, une des deux mains doit ensuite être ouverte à plat. La main restante permet alors d'orienter librement le volume de sélection dans l'espace et désigner les objets d'intérêt. Enfin, en ouvrant la seconde main, la sélection devient effective. La figure 5.14 montre le fonctionnement du lasso $3D$.

Nous utilisons la technique du lasso $3D$ pour remplacer la technique de sélection par volume parallélépipédique lorsque l'on souhaite sélectionner un groupe de sommets d'un maillage. En effet, le lasso $3D$ s'avère plus précis, car il peut être tourné dans l'espace et sa forme peut changer tant que la main

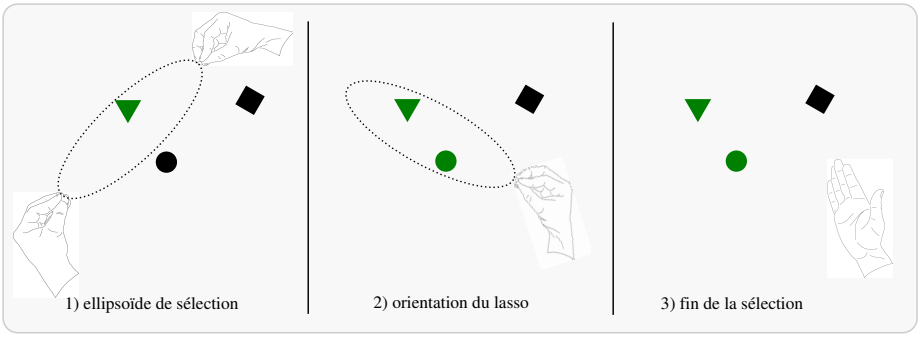


Fig. 5.14: Technique de sélection mutliple par lasso $3D$ à l'aide conjointe des deux mains.

non-dominante demeure fermée.

5.6 Conclusion

Dans ce chapitre nous avons présenté notre travail sur l'interaction $3D$. Nous avons d'abord introduit le concept d'interaction conduite par la dimension qui permet de concevoir avec plus d'exactitude le couple outil d'interaction – objet dont une application $3D$ peut avoir besoin. Notre apport se situe à la description de règles de conception que tout programmeur devrait mettre en œuvre en amont de l'écriture de son programme, et également à la rédaction des 3 lois de l'interaction qui permettent de s'assurer que la chaîne d'interaction imaginée peut effectivement être réalisée selon le matériel dont disposera l'utilisateur.

Notre approche de l'interaction nous a amené à nous intéresser à la reconnaissance des postures et des gestes. En effet, l'utilisation conjointe des deux mains permet de supprimer en partie le besoin d'un périphérique physique, tout en augmentant, dans le même laps de temps, les performances de l'interaction. Celle-ci est plus naturelle, car la chaîne d'interaction est plus courte. Nous avons voulu nous pencher sur l'ensemble de problèmes relatifs à ce domaine. Cela nous a amené à coupler deux algorithmes – ICP et Kalman – dans le but avoué de stabiliser le suivi des mouvements des mains, et le rendre aussi cohérent que possible. En associant à ceci une méthode de calibrage simple et deux méthodes de reconnaissance de postures – méthode brute et méthode par réseau de neurones, la librairie vrLib offre à présent des mécanismes gestuels à tout programmeur d'application $3D$. Nous avons également imaginé une méthode de reconnaissance de gestes originale, basée sur le concept de boussole $3D$ et d'un automate à états.

Afin de tester notre reconnaissance de postures, nous avons proposé trois métaphores d'interaction spécifiquement $3D$. La première, le rayon déformable permet de définir un chemin de navigation quelconque autorisant les déplacements contraints dans la scène. Cette technique sert également à sélectionner des objets partiellement cachés, et une perspective de manipulation contrainte est envisagée. La seconde, le volume de sélection parallélépipédique se veut une technique très simple et très rapide d'utilisation, non orientable, pour la sélection

multiple d'objets. Enfin, la troisième et dernière métaphore de forme ellipsoïdale, dénommée le lasso $3D$, est un peu plus complexe à employer. L'utilisation conjointe des deux mains sert à définir en une passe le volume et son orientation.

Chapitre 6

Applications de test de la vrLib

6.1 Introduction

Le chapitre précédent couvre la vrLib, la librairie que nous avons mise au point pour le développement rapide et efficace d'applications de réalité virtuelle. Le chapitre que nous présentons ici introduit différentes applications que nous avons choisi de construire à l'aide de notre boîte à outils vrLib.

Nous allons commencer par présenter les deux applications principales sur lesquelles nous avons choisi de concentrer nos efforts, tant en terme d'interaction que d'interface utilisateur. La première, qui se nomme MRMaps^{VR}, est un logiciel de modélisation géométrique à base de cartes multi-résolution [KCB06]. Il s'agit d'un laboratoire d'expérimentation qui nous sert de support à la comparaison d'une application $2D$ portée en environnement virtuel. La seconde application, intitulée Cad^{VR}, est plus particulièrement consacrée à l'interaction. Dans le cadre de la thèse d'Arnaud Fabre [Fab06] sur les constructions géométriques et la résolution de contraintes $3D$, nous testons différentes solutions de pose de contraintes $3D$ à l'aide de gestes.

Dans une seconde partie, nous présentons deux autres applications réalisées par des étudiants de Master, dans le cadre de leur cursus universitaire. Le premier logiciel est appelé STIGMA^{VR} et dont le but est de proposer un modeleur d'animation à base d'objets $4D$. La seconde application s'intitule SoundShaker^{VR} et est destinée à manipuler graphiquement des paramètres d'objets sonores dans le cadre d'un concert. D'autres applications géométriques sont actuellement en cours de portage à l'aide de la vrLib au sein de notre laboratoire.

6.2 Le modeleur multi-résolution MRMaps^{VR}

L'application que nous présentons ici permet de manipuler des surfaces de subdivision multi-résolution [Zor05]. Nous allons commencer par introduire les notions relatives à ces dernières, puis nous présenterons brièvement le modèle

que nous utilisons pour le représentation de ces surfaces. Ensuite, nous montrerons avec quelle facilité nous avons porté une première version de notre modeleur multi-résolution en environnement immersif *via* la vrLib. Puis, nous expliquerons comment nous avons amélioré l'interaction $3D$ pour améliorer le travail de l'utilisateur, en rendant les outils plus efficaces et plus intuitifs. La figure 6.1 donne l'architecture globale de ce logiciel.

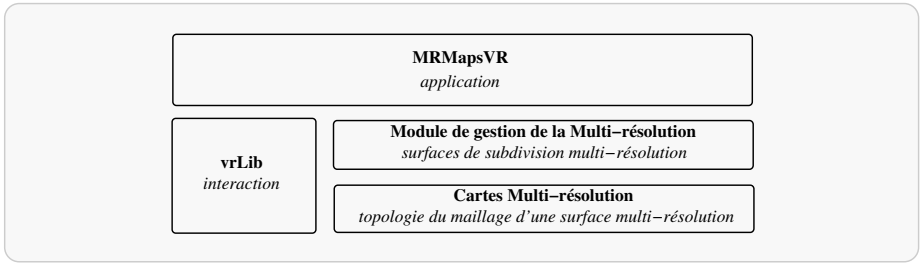


Fig. 6.1: Architecture de l'application MRMaps^{VR}.

6.2.1 Description de l'application

Cette section donne une description de MRMaps^{VR}, en précisant le type d'objets qui est manipulé, ainsi que la manière de les représenter. Nous introduirons les surfaces de subidivision, puis leur extension multi-résolution. Enfin, nous discuterons des 2-cartes multi-résolution auxquelles est associé un plongement permettant leur représentation géométrique.

Les surfaces de subdivision multi-résolution

Surfaces de subdivision Le principe des surfaces de subdivision est de définir une surface comme la limite à l'infini d'une séquence de maillages raffinés successivement. Un schéma de subdivision définit la manière d'appliquer ce raffinement au maillage. Nous pouvons distinguer ici deux étapes : d'abord, la topologie du maillage est raffinée, puis de nouvelles informations géométriques sont calculées à partir du maillage de départ et associées au maillage obtenu. Différents schémas génèrent des surfaces aux propriétés différentes.

Le processus de **raffinement de la topologie** d'un schéma de subdivision est habituellement classé en deux familles, appelées **primale** et **duale**. Dans un schéma de subdivision primal, ce sont les faces du maillage qui sont subdivisées. Ceci est réalisé en coupant leurs arêtes et en reliant les sommets ainsi créés afin de former de nouvelles faces. Les schémas de Loop [Loo87], de Catmull-Clark [CC78] ou le schéma Butterfly [DLG90] sont des exemples de schémas primaux. Ceux-ci travaillent sur des maillages de type différents : le schéma de Loop ou le Butterfly sont basés sur des maillages triangulaires, celui de Catmull-Clark sur des maillages quelconques (après un pas de subdivision, toutes les faces sont des carrés). Dans un schéma de subdivision dual, ce sont les sommets du maillage qui sont subdivisés. Les faces du maillage de départ sont réduites, et les trous ainsi créés sont comblés avec de nouvelles faces. Les schémas de Doo-Sabin [DS78] ou le schéma MidEdge [PR97] sont des exemples de schémas duaux. Ceux-ci

travaillent sur des maillages quelconques.

Le processus de **calcul de la géométrie** d'un schéma de subdivision est également classé en deux familles, appelées **interpolante** et **approximante**. Dans les deux cas, les informations géométriques associées aux sommets sont calculées localement pour chaque sommet en appliquant un masque sur les positions des sommets voisins dans le maillage de départ. Dans un schéma approximant, les positions de tous les sommets du nouveau maillage sont calculées et se rapprochent de la surface limite. Dans un schéma interpolant, les sommets du maillage de départ se situent déjà sur la surface limite, et seules les positions des nouveaux sommets insérés sont calculées.

Multi-résolution Le principe de l'extension multi-résolution des surfaces de subdivision est de **conserver tous les maillages intermédiaires** du processus de subdivision – appelés alors niveaux de résolution – et d'introduire des **vecteurs de détail** entre ces niveaux. Ces vecteurs expriment la différence en chaque sommet entre sa position au niveau i et celle résultant de la subdivision du niveau $i-1$.

Une forte connexion existe entre les surfaces multi-résolution et les ondelettes, et en particulier les deux opérations d'analyse et de synthèse. L'analyse calcule les positions des sommets du maillage de niveau $i-1$ en appliquant un filtre moyenneur sur le maillage de niveau i et calcule par la même occasion les vecteurs de détail. La synthèse reconstruit les données au niveau i en subdivisant le maillage de niveau $i-1$ et en y ajoutant les vecteurs de détail.

Deux approches existent pour générer des surfaces de subdivision multi-résolution : l'approche "grossier vers fin" et l'approche "fin vers grossier". La première démarre d'un objet grossier (qui constituera le niveau de résolution minimum de l'objet), et construit les niveaux fins en appliquant un schéma de subdivision. La deuxième part d'un objet fin (qui constituera le niveau de résolution maximum de l'objet) et construit les niveaux grossiers en appliquant le processus d'analyse – dans ce cas, le maillage de départ doit avoir une connectivité de subdivision, *i.e.* la même connectivité que s'il avait été généré par un schéma de subdivision.

Une fois un tel objet construit, le **maillage peut être édité à des niveaux de résolution différents** : une édition à un **niveau grossier** entraîne une **déformation globale**, en conservant les informations fines grâce aux vecteurs de détail ; une édition à un **niveau fin** entraîne une **déformation plus locale** de l'objet. À chaque édition les niveaux plus fins sont mis à jour par le processus de synthèse, et les niveaux plus grossiers par le processus d'analyse.

Modèles de représentation des objets

Le modèle que nous utilisons pour la représentation des surfaces de subdivision multi-résolution est celui des **2-cartes multi-résolution**. Ce modèle est défini comme une extension des 2-cartes. Nous allons commencer par rappeler ce que sont les 2-cartes, puis présenter leur extension multi-résolution.

Une 2-carte permet de décrire la topologie de 2-variétés fermées orientables, c'est-à-dire de surfaces englobant des volumes. Cette topologie est décrite en subdivisant les objets en cellules de différentes dimensions (sommets, arêtes, faces) partageant des relations d'incidence et d'adjacence. Une 2-carte est constituée d'un ensemble de brins, reliés entre eux par des relations. Formellement, étant donné un ensemble fini de brins B, une involution sans point fixe α_0 et une permutation α_1 sur cet ensemble B, une 2-carte est un triplet :

$$M = (B, \alpha_0, \alpha_1)$$

.

L'extension des 2-cartes pour la représentation des surfaces de subdivision multi-résolution permet de décrire la topologie du maillage de surfaces multi-résolution. Chaque niveau de résolution est représenté par une 2-carte. Cet ensemble de 2-cartes s'imbrique naturellement, les brins décrivant un niveau donné étant réutilisés dans les niveaux plus fins, tout en conservant les relations topologiques entre chaque niveau.

Plus formellement, une 2-carte multi-résolution est composée d'un ensemble de brins B et de relations entres les éléments de cet ensemble. Les brins de B sont ici indexés par le niveau de résolution auquel ils sont introduits. Les brins décrivant l'objet au niveau le plus grossier appartiennent à l'ensemble B^0. Des brins sont ajoutés à cet ensemble pour décrire le maillage de niveau supérieur (au sens "plus fin") et former l'ensemble B^1. Pour décrire chaque niveau de résolution supérieur, des brins sont ajoutés à ceux décrivant le maillage de niveau inférieur, constituant ainsi une imbrication d'ensembles $B_0 \in ... \in B_n = B$. Les relations entre les brins sont paramétrées par le niveau de résolution, ainsi une 2-carte multi-résolution est un triplet :

$$M = (B, \{\alpha_0^i\}_{i\in[0..n]}, \{\alpha_1^i\}_{i\in[0..n]})$$

tel que chaque niveau de résolution est défini par la 2-carte :

$$M^i = (B^i, \alpha_0^i, \alpha_1^i)$$

.

Un tel modèle ne décrit que la topologie de la subdivision des objets. Il faut donc définir un modèle de plongement décrivant les données géométriques, que l'on va associer à chaque cellule du maillage, pour représenter complètement un objet. Le plongement le plus simple est le 0-plongement, où seules les sommets sont plongés – c'est-à-dire que l'on associe un point $3D$ à chaque sommet du maillage. Le plongement des autres cellules est obtenu par interpolation linéaire du plongement des sommets. Des modèles de plongement plus évolués peuvent bien sûr être utilisés, associant par exemple des courbes aux arêtes ou des carreaux de surface aux faces. Le détail de la formalisation des 2-cartes multi-résolution et de leur plongement est donné dans Kraemer *et al.* [KCB06].

6.2.2 Interface et interaction

MRMaps^{VR} est une application dont les constituants graphiques sont les sommets, les arêtes et les faces. L'utilisateur doit être en mesure de les manipuler, au même titre qu'il doit pouvoir interagir le plus simplement possible avec

le modèle topologique des cartes multi-résolution. Pour bien faire, le modèle de représentation des surfaces de subdivision multi-résolution devrait être "transparent" pour l'utilisateur ; il ne devrait pas se soucier du contrôle du modèle et pouvoir se concentrer pleinement sur la modélisation géométrique.

Les travaux de recherche de Pierre Kraemer, David Cazier et Dominique Bechmann sur les applications de la multi-résolution à la modélisation géométrique ont abouti à une première application $2D$: MRMaps. Celle-ci permet de manipuler basiquement des surfaces de subdivision multi-résolution. Nous allons voir que le portage de l'application $2D$ en $3D$ peut se faire rapidement et simplement à l'aide de la vrLib. Ensuite, nous montrerons que l'évolution de l'interaction $3D$ permet d'améliorer l'ergonomie de l'application et les capacités de l'utilisateur. Enfin, nous expliquerons comment le modèle de déformation DOGME peut servir à augmenter les possibilités d'édition des objets multi-résolution, de manière à parfaire l'interaction.

De l'interface $2D$ à l'interface $3D$

Nous avons souhaité porter la version $2D$ de l'application MRMaps, écrite au moyen de la librairie Qt de la société Trolltech, en environnement immersif à l'aide de la vrLib. La figure 6.2 donne un aperçu de ce logiciel qui se manipule à l'aide d'une souris et d'un clavier.

La vrLib nous offre tous les outils nécessaires à la ré-écriture de l'interface et de l'interaction $3D$, très simplement et rapidement. Bien que la qualité d'un programme soit partiellement dépendante de la façon de coder, les versions Qt et vrLib ont une taille similaire, de l'ordre de 1500 lignes chacune. Ce chiffre ne porte que sur l'interface et l'interaction.

Concrètement, le programmeur peut se consacrer à développer les fonctionnalités liées à l'édition des surfaces de subdivisions multi-résolution sans avoir à se soucier de la gestion du matériel et des problèmes d'interface et d'outil d'interaction. MRMaps^{VR} propose une interface comparable à la version $2D$: les widgets sont identiques, bien qu'ils soient affichés en $3D$, l'interaction est réalisée *via* la technique du lancer de rayon. La figure 6.3 illustre l'interface de l'application $3D$. MRMaps^{VR} est développé sur un Workbench, l'espace de travail et l'espace virtuel de modélisation sont quasiment confondus, et la manipulation des objets est directe.

Le passage du pointeur de souris au lancer de rayon a pour avantage de conserver une interaction simple à mettre en œuvre, cependant, il ne s'agit pas d'une technique très intuitive pour l'utilisateur. Celui-ci ne profite pas pleinement des capacités offertes par le matériel, comme par exemple les gants de données. La manipulation des sommets, arêtes et faces des objets géométriques est réalisée à l'aide de la technique du lancer de rayon ; en appuyant sur un bouton du dispositif physique, l'utilisateur débute la manipulation sur les objets qui se trouvent pris dans le rayon. Les changements de point de vue, les translations, les rotations et les changements d'échelle sont réalisés à l'aide de la même technique d'interaction mais d'un bouton différent. Le manipulateur contrôle alors l'ensemble des objets géométriques, en manipulant directement la scène

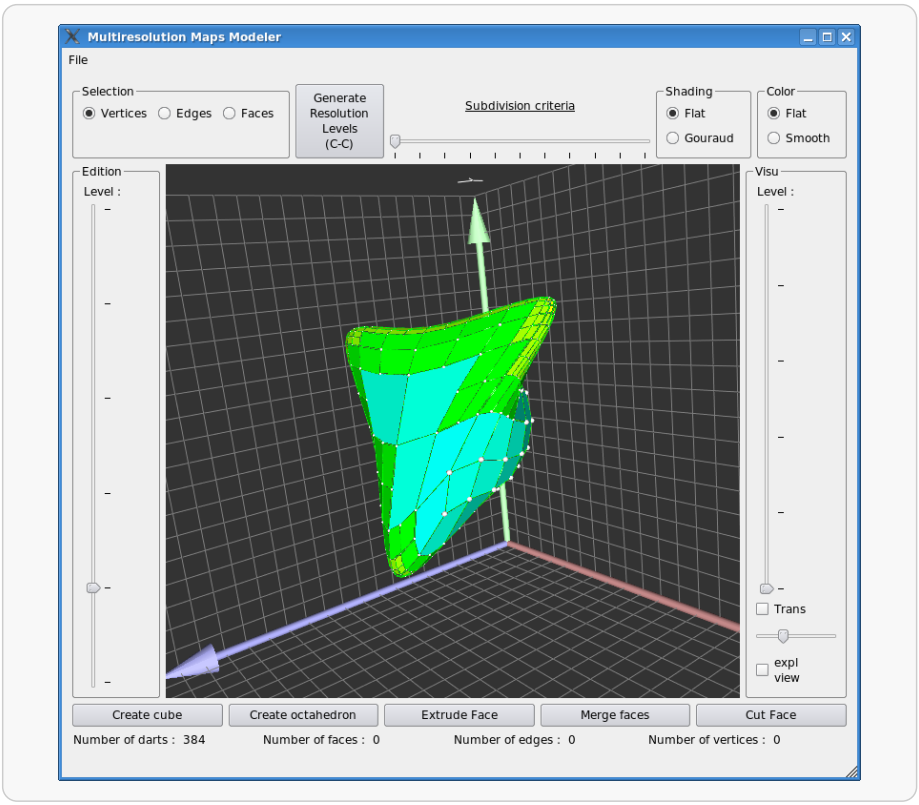


Fig. 6.2: Interface $2D$ de l'application MRMaps^{VR}.

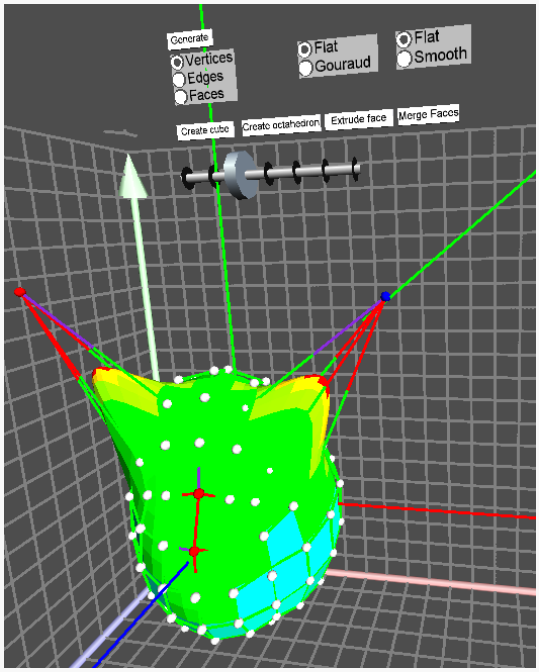


Fig. 6.3: Interface $3D$ de l'application MRMaps^{VR}.

qui les contient, à la manière du Scaled-world grab de Mine *et al.* [MBJS97].

Évolution de l'interaction $3D$

Nous venons de montrer que la vrLib permet certes d'obtenir une première version $3D$ de l'application MRMaps^{VR}, mais au sein de laquelle l'interaction reste assez limitée. Nous souhaitons faire évoluer cette dernière pour que l'utilisateur dispose de conditions de travail optimales. Nous proposons de remplacer le périphérique de manipulation du rayon virtuel, par une interaction gestuelle reposant sur les gants de données et la reconnaissance de postures.

Les widgets de contrôle de l'application sont conservés, mais ils sont déplacés à plat sur l'écran du bas de notre dispositif d'immersion, le *Workbench*. Les gants de données que nous avons à notre disposition sont munis de capteurs de pression placés sur le bout des phalanges. À l'aide de ces derniers, l'utilisateur peut contrôler l'interface directement à la surface de l'écran. Cette surface de contact permet de contraindre physiquement l'interaction. L'utilisateur est ainsi plus précis et plus rapide dans la manipulation des widgets.

Plutôt que de sélectionner les sommets, les arêtes et les faces par pointage à l'aide du lancer de rayon, l'utilisateur emploie ses propres mains pour manipuler les cellules. La tâche de sélection se déroule de la façon suivante. Un pincement de la main dominante sert à sélectionner – resp. désélectionner – un point non sélectionné – resp. déjà sélectionné. Puis, pour déplacer l'ensemble des points

sélectionnés, l'utilisateur ferme le poing et le bouge dans l'espace. Il en est de même pour les arêtes et les faces. Le déplacement des objets est identique celui de la main dominante. Cette approche est plus intuitive et rapide que la technique du lancer de rayon, et il n'y a pas d'intermédiaire entre la main et les objets virtuels.

Lorsque le nombre de cellules à manipuler est important, les métaphores de sélection par volume vues au chapitre 2 de la partie 2 sont conseillées. L'utilisateur définit le volume à l'aide de ses deux mains *via* une posture de pincement. Lorsque certains objets non souhaités tombent par inadvertance dans le volume de sélection, l'utilisateur peut les désélectionner les uns après les autres par un simple pincement de l'index et du pouce.

La première version de MRMaps^{VR} ne proposait qu'un mode de navigation limité. La surface de contact avec le *Workbench* est très utile pour cette tâche d'interaction.

D'une part, le capteur de pression de l'index de la main non dominante est utilisé pour rapprocher temporairement – effet de zoom – l'objet multi-résolution en cours d'édition. De cette façon, il est possible d'observer sa surface avant de la modifier, et l'objet reprend sa position initiale lorsque le capteur n'est plus en contact avec la surface.

D'autre part, nous souhaitons qu'un objet puisse être tourné sur lui-même afin de pouvoir l'étudier sous un autre angle. La technique que nous proposons repose sur l'utilisation d'une zone de contact rectangulaire sur l'écran du bas, et qui fonctionne comme un tapis de souris virtuel. Les capteurs de pression de l'index et du majeur sont employés pour manipuler le tapis virtuel à la manière d'un trackball. Les déplacements de la main sur le tapis sur les axes X et Y permettent de tourner l'objet sur lui-même de façon très précise.

Ainsi, les déplacements des objets multi-résolution sont entièrement réalisés à l'aide des deux mains et de la surface de contact qui est offerte par l'écran du bas du *Workbench*. L'utilisateur contrôle plus efficacement les objets, en exploitant mieux le matériel dont il dispose.

6.2.3 Association d'un modèle de déformation à l'édition multi-résolution grâce à l'interaction $3D$

L'édition multi-résolution consiste à éditer un niveau de résolution donné ; ce niveau est édité soit de façon très simple avec les outils que nous venons de voir, soit avec des outils plus évolués comme celui que nous allons présenter maintenant. Il s'agit du modèle de déformation DOGME.

Introduction au modèle de déformation DOGME

Dans les paragraphes précédents, nous avons vu que l'utilisateur devait sélectionner lui-même les cellules de la 2-carte, pour un niveau de résolution donné, avant d'être en mesure de les manipuler. En confiant la sélection à un modèle de déformation, l'utilisateur réalise les tâches de sélection et de manipulation

en une fois. Il ne travaille plus directement sur la surface de subdivision multi-résolution, mais simplement en définissant une contrainte et un volume de déformation. Le modèle de déformation retenu est DOGME de Borrel et Bechmann [BB91].

Plutôt que de manipuler, pour un niveau de résolution donné, certaines cellules de la 2-carte, et donc avoir à les sélectionner préalablement, nous confions cette tâche à DOGME. La déformation est un objet défini par le déplacement de points appelés **contraintes**. Nous utilisons ce modèle dans sa version la plus simple, c'est-à-dire en définissant une contrainte linéaire à l'aide de deux points de contrainte et d'un **volume d'influence**, comme illustré sur la figure 6.4. Ce dernier permet de stipuler à DOGME quels sont les points affectés par la déformation. Une certaine portion de l'espace est affectée par la contrainte, selon la formulation d'une fonction mathématique appelée **fonction d'extrusion**. En entrée, DOGME manipule des points et ne considère aucunement la topologie qui y est éventuellement associée ; puis, après définition d'une ou plusieurs contraintes, en ressort un nuage de points déformés. La figure 6.5 montre l'exemple d'un maillage plan déformé par trois contraintes linéaires.

DOGME est caractérisé par la fonction de déformation suivante :

$$D(X) = X + M \cdot F(X) = \tilde{X},$$

où F est une fonction ($F : R^n \rightarrow R^m$) qui spécifie comment la densité de la déformation agit autour de chaque contrainte et M est une matrice, de taille $n \times m$, encapsulant une transformation linéaire combinant l'interaction de toutes les contraintes.

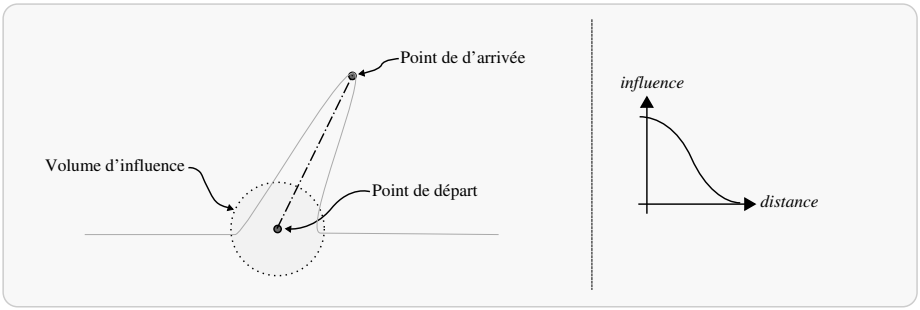


Fig. 6.4: Modèle de déformation DOGME.

Manipulation du modèle en interaction $3D$ à l'aide de gestes

L'utilisateur ne manipule plus directement les cellules de la surface de subdivision multi-résolution, en les sélectionnant directement. DOGME prend en charge les opérations de sélection et de déplacement des sommets de la surface. Pour cela, il faut préalablement définir deux points de contrainte – départ et arrivée, ainsi qu'un volume d'influence.

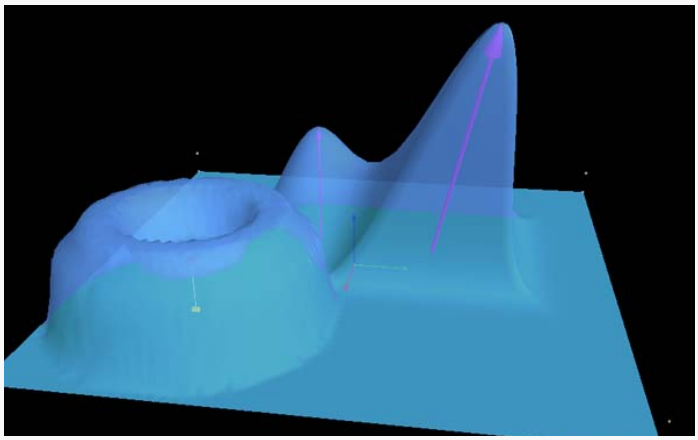


Fig. 6.5: Exemple de déformation avec le modèle de déformation DOGME. Un plan est déformé *via* la pose de 3 contraintes linéaires.

Le volume d'influence est sphérique, quelle que soit la fonction d'extrusion utilisée. Il se manipule à l'aide conjointe des deux mains, tout comme le reste de la déformation. Le point de départ de la contrainte est posé en fermant le poing de la main dominante. Tant que le poing reste fermé, on édite à la fois la contrainte et on observe le résultat possible de la déformation.

La main dominante permet de poser le point de départ en fermant le poing. Puis le point de départ est attaché à la main dominante, jusqu'à ce que l'utilisateur décide de refermer à nouveau le poing. La contrainte est alors calculée et appliquée définitivement à la 2-carte au niveau de résolution courant. La distance entre le point de départ et la main non dominante détermine le rayon du volume d'influence. On termine de poser la contrainte en relâchant le poing, qui est alors appliquée définitivement au nuage de points. La distance entre le point de départ et la main non dominante détermine le rayon du volume d'influence.

Application du modèle de déformation à l'édition multi-résolution

La procédure d'édition que nous venons de décrire est appliquée dans le cadre de l'édition multi-résolution. La déformation est appliquée aux sommets du maillage du niveau de résolution courant. La figure 6.6 illustre le résultat de trois contraintes linéaires posées à l'aide des deux mains sur une surface de subdivision *via* le modèle de déformation DOGME.

6.2.4 Conclusions et perspectives

Nous venons de discuter du portage de l'application MRMaps en environnement virtuel. Celle-ci permet de mettre en évidence que la librairie vrLib facilite le portage ou l'écriture d'une application $2D$ en environnement $3D$ en proposant des outils d'interaction simples, des widgets et des fonctions de conception d'une interface graphique simples à mettre en œuvre et efficaces. Plusieurs versions de

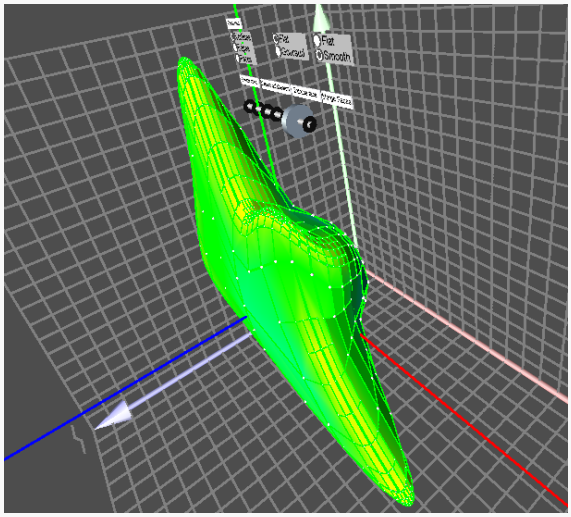


Fig. 6.6: Modélisation multi-résolution à l'aide du modèle de déformation DOGME.

l'application ont succédées à la version $2D$. Dans un premier temps, nous avons réalisé une correspondance presque complète de l'interface utilisateur entre la version sur station et la version sur le *Workbench*, biens que la nature du périphérique soit différente.

Puis, dans une seconde version du logiciel, nous avons remplacé et amélioré les modalités d'interaction de la technique du lancer de rayon. L'utilisation conjointe des deux mains et de la reconnaissance de postures, associée aux capteurs de pression des gants de données et de la surface de contact du *Workbench*, permet de parfaire l'interaction et l'immersion de l'utilisateur. Les techniques de sélection par volume, comme le lasso $3D$, apportent une amélioration notable du confort de l'utilisateur. Cependant, ces dernières ne permettent pas de contrôler à la fois rapidement, et surtout finement, un ensemble important de sommets.

Enfin, dans la troisième et dernière version de l'application nous avons cherché à améliorer l'ergonomie de l'interface utilisateur. D'une part, les widgets qui composent l'interface utilisateur sont affichés sur la surface du bas de l'écran du *Workbench* afin de rendre leur contrôle plus efficace et plus précis. D'autre part, l'utilisation d'un modèle de déformation tel que DOGME nous a permis d'introduire une réponse originale, qui permet de réaliser sélection et manipulation simultanément. Les contraintes et le volume d'influence sont définis à l'aide des deux mains de l'utilisateur. Les résultats sur la manipulation des surfaces de subdivision multi-résolution sont très encourageants. Nous explorons également l'utilisation d'un autre modèle, Twister de Llamas *et al.* [LKG+03] qui permet de déformer un ensemble de points selon une hélicoïde.

6.3 Le modeleur géométrique Cad^{VR}

La seconde application que nous présentons est Cad^{VR}. Il s'agit d'une application permettant la pose de contraintes $3D$ simples. Cette application a été écrite à l'aide de vrLib et illustre la simplicité et l'efficacité de l'utilisation des gants de données et de la reconnaissance de postures pour la manipulation des constructions géométriques en $3D$. La figure 6.7 illustre l'architecture globale de ce programme.

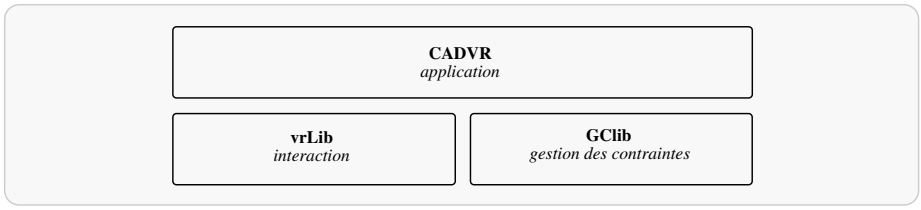


Fig. 6.7: Architecture de l'application Cad^{VR}.

6.3.1 Introduction

Bon nombre de logiciels dédiés à la géométrie $2D$ [Mec, Jac, Kor99, Cha99, TAC00] ont été conçus pour le domaine de l'éducation assistée par ordinateur ou EAO[1]. Ils définissent tous un espace de travail dans lequel des outils classiques, comme la règle et le compas, sont adaptés à une représentation iconique sur ordinateur, améliorent les capacités du couple stylo – papier utilisé traditionnellement. En effet, puisque l'utilisateur est capable de dessiner des points et des lignes sur un écran d'ordinateur, il est très facile de modifier la figure géométrique par manipulation directe *via* la souris, tout en conservant ses propriétés vérifiées. C'est le fonctionnement de Cabri-géomètre [Bau91, BM92], une des applications la plus utilisée de constructions géométriques $2D$, défini sous le terme de *Géométrie Dynamique*.

Cependant, la troisième dimension a été faiblement étudiée. Quelques logiciels comme Cabri $3D$ [Qas97], Geospacw [Geo], ou encore Calques $3D$ [VL98, VL99] peuvent être cités. Mais, leurs répercussions dans le domaine de l'apprentissage de la géométrie $3D$ n'ont pas été concluantes. Dans la pratique, la visualisation et la manipulation d'objets géométriques $3D$ avec des outils $2D$ ne sont pas suffisamment intuitifs et efficaces. En fait, le principal problème des applications géométriques dans l'espace est le fossé qui existe entre un concept d'objet et sa représentation. L'utilisation d'un environnement virtuel améliore la compréhension des constructions géométriques $3D$ puisque l'utilisateur est capable d'atteindre intuitivement les différentes entités géométriques et d'interagir avec elles.

[FMS04] ont décrit un prototype de constructions géométriques $3D$ en environnement virtuel, nommé Coyote-géomètre, lequel s'apparente à Cabri-géomètre [Bau91]. Notre approche est basée sur l'utilisation des gants de données pour

[1] a. *Computer Assisted Education* ou *CAE*

reconnaître des gestes et des postures. C'est la principale différence avec des études précédentes comme Construct3D de Hannes Keufmann [Kau02]. Appliquée à un autre domaine, notre approche est voisine de celle de Zeleznik *et al.* [ZHH96] qui propose SKETCH, une interface basée sur les gestes pour « approximer » la modélisation polyédrique $3D$.

Coyote consiste à fournir une meilleure perception et compréhension d'un espace géométrique. Mais, notre expérience a prouvé qu'il est inefficace de donner trop de possibilités d'interaction à l'utilisateur final puisque la complexité inhérente à la géométrie en 3 dimensions nécessite qu'il soit guidé dans le processus de construction. L'interaction gestuelle est une approche très intuitive et naturelle, bien que ses nombreux degrés de liberté ne soient pas faciles à appréhender. Il est donc important de trouver des solutions pour contraindre l'interaction. Nous pensons que l'utilisation d'une interaction gestuelle contrainte permettra de réduire la complexité induite par la combinaison de la réalité virtuelle et des constructions géométriques $3D$.

Nous allons commencer par donner un exemple de construction, puis nous montrerons comment créer et sélectionner des objets dans une scène géométrique en utilisant une interaction gestuelle. Ensuite le processus de construction bimanuel sera exposé. Dans la section suivante, nous exposerons le processus de manipulation de la figure résultante. Enfin, la navigation contrainte à l'aide de notre métaphore de rayon déformable sera décrite.

6.3.2 Exemple de construction

Considérons un univers géométrique composé de 6 types d'objets : points, segments, lignes, plans, cercles, et sphères, ainsi que 30 primitives de construction, comme construire une ligne passant par deux points, etc. Bien qu'il n'y ait que peu d'objets différents, cela est suffisant pour manipuler un large choix de primitives et pour une utilisation intéressante en EAO. Nous aimerions maintenant vérifier que *l'intersection entre un cube et un plan donne au plus un hexagone*. Puisqu'il n'y a pas d'objet « cube » dans notre univers, nous devons le construire.

Un plan de construction d'un cube pourrait être le suivant (il y a généralement plus d'un plan de construction correspondant à la solution d'un problème) :

1. soit p_1 un point ;
2. soit d_1 une ligne passant par p_1 ;
3. soit d_2 une ligne perpendiculaire à d_1 passant par p_1 ;
4. construisons π comme le plan passant par d_1 et d_2 ;
5. soit p_2 un point de d_1 ;
6. construisons σ la sphère ayant pour centre p_1 passant par p_2 ;
7. soit p_3 une des intersections de d_2 et σ ;
8. soit d_3 une ligne perpendiculaire au plan π passant par p_1 ;
9. soit p_4 une des intersections de d_3 et σ ;
10. construisons la ligne parallèle d_4 à d_1 passant par p_3 ;

11. construisons la ligne parallèle d_5 à d_2 passant par p_2 ;
12. soit p_5 l'intersection de d_4 et d_5 ;
13. construisons la ligne parallèle d_6 à d_1 passant par p_4 ;
14. construisons la ligne parallèle d_7 à d_3 passant par p_2 ;
15. soit p_6 l'intersection de d_6 et d_7 ;
16. construisons la ligne parallèle d_8 à d_4 passant par p_3 ;
17. construisons la ligne parallèle d_9 à d_2 passant par p_2 ;
18. soit p_7 l'intersection de d_8 et d_9 ;
19. construisons la ligne parallèle d_10 à d_6 passant par p_6 ;
20. construisons la ligne parallèle d_11 à d_4 passant par p_5 ;
21. soit p_8 l'intersection de d_10 et d_11.

Puis, un plan libre est créé et les intersections entre le plan et le cube sont calculées durant la manipulation de ces deux objets. Cet exemple montre que quelques objets sont libres, comme p_1, alors que d'autres sont à moitié définis comme p_2, et d'autres encore complètement définis comme p_5. Ainsi la sélection, la création et la manipulation sont différentes pour chaque type d'objet. Il montre également qu'une construction géométrique peut être décomposée en deux étapes : le choix de la construction à réaliser, et la sélection des objets impliqués. Enfin, une navigation, dite pédagogique, est nécessaire pour réellement voir comment obtenir un hexagone comme celui de la figure 6.8. Nous nous servirons de l'exemple précédemment décrit dans les sections suivantes.

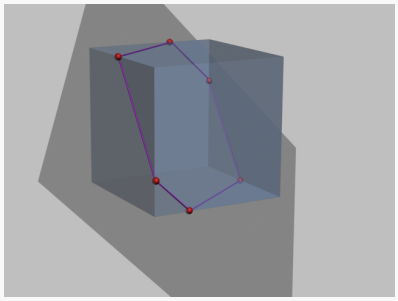


Fig. 6.8: Intersection entre un cube et un plan.

6.3.3 Création et sélection bi-manuelle

Réaliser des constructions dynamiques $3D$ n'est pas très simple. Un système d'affichage stéréoscopique, dans notre cas le *Workbench* pourrait aider l'utilisateur dans cette tâche. Un ensemble de caméras infrarouges permet de suivre à la fois les mouvements de la tête et ceux des mains en suivant les positions et les orientations. En portant une paire de gants de données disposant de récepteurs infrarouges fixés sur le dos de la main, il est possible d'effectuer des gestes et réaliser une construction.

En démarrant une nouvelle construction, l'environnement du *Workbench* est visuellement vide. L'utilisateur peut ajouter librement des objets géométriques, ce qui correspond à l'étape de création. Puis, il peut les désigner pour les manipuler : ce sont les étapes de sélection et manipulation. Les changements de mode entre création, sélection et manipulation sont automatiques. Pour la navigation en revanche, l'utilisateur doit préciser qu'il souhaite changer de mode.

Le mode création commence en pointant un endroit vide de la scène avec la main dominante, et en fermant la main non dominante dans le même temps. Un nouvel objet est ajouté et affiché dans l'environnement virtuel, selon la forme de la main dominante de l'utilisateur (figure 6.9).

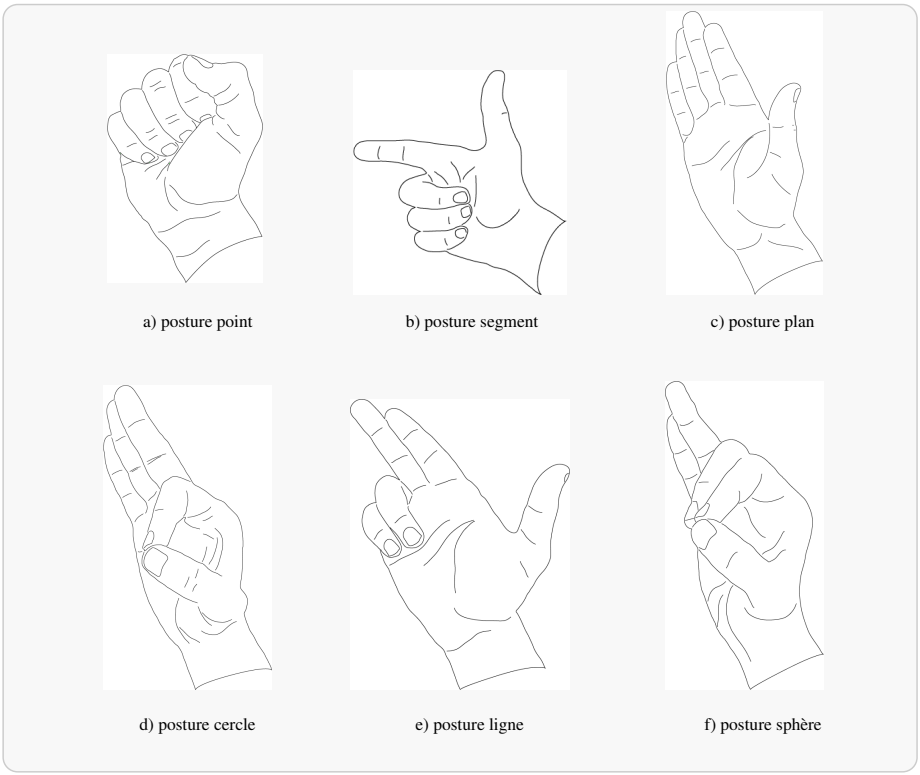


Fig. 6.9: Dictionnaire de gestes.

Création d'objets géométriques

Il y a bien trop d'objets différents en constructions géométriques et leurs combinaisons sont trop nombreuses pour avoir une unique métaphore pour interagir sur eux. Lorsque plusieurs techniques sont employées en même temps, chacune devrait correspondre visuellement à un objet précis, et ainsi agir comme un outil-icône dans l'esprit. Ainsi, l'ensemble de nos gestes forme une espèce de vocabulaire. Chaque posture a une signification visuelle : par exemple, une main à plat sert à ajouter un plan dans la scène (figure 6.9.c). Il s'agit donc d'un langage de postures dont la sémantique d'une posture est associée à la sémantique

d'une commande et agit comme un raccourci sémantique. Le dictionnaire de gestes, figure 6.9 montre cette association.

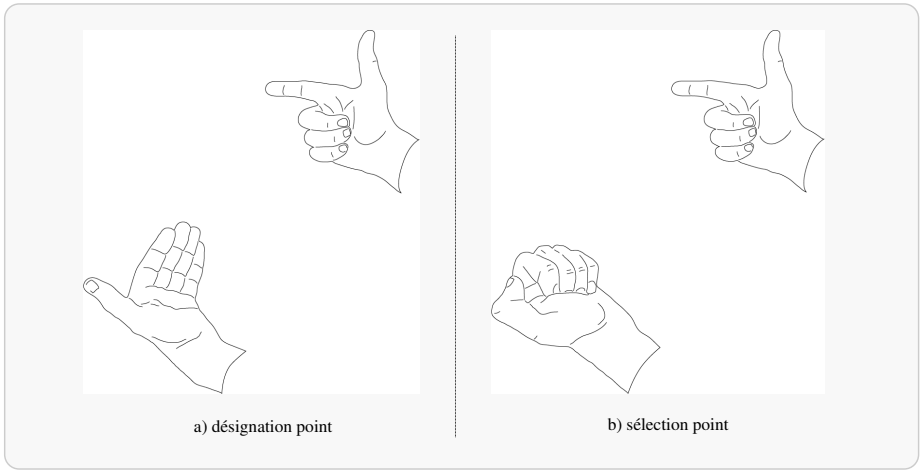


Fig. 6.10: Désignation et sélection d'un point.

Puisque nous disposons de peu de primitives géométriques, nous avons choisi de créer un dictionnaire avec une posture par entité. Chacun a été choisi pour être suffisamment éloigné des autres, et ainsi éviter des ambiguïtés. À l'aide des gants NoDNA X-ist, le système peut déterminer les mesures des angles au niveau des phalanges, ainsi que la position spatiale de la main. Ces valeurs sont ensuite interprétées pour reconnaître les postures, selon une méthode brute ou une méthode s'appuyant sur un réseau de neurones par gant. Avec l'approche par intelligence artificielle, la première phase consiste à initialiser le système et familiariser le sujet avec le système. Puis, les réseaux sont entraînés en collectant les données issues des gestes de l'utilisateur. Le système enregistre un profil par utilisateur et tous les profils sont ensuite combinés pour proposer un réseau par défaut pour un nouveau sujet.

Sélection

La sélection se décompose en deux phases comme l'illustre la figure 6.10. La désignation est la première, durant laquelle l'utilisateur pointe un objet, main non dominante ouverte. Puis, dans la seconde phase, il valide la sélection en fermant cette main (posture du poing). L'application propose le principe des calques qui permettent de décomposer le processus de construction en cachant les étapes intermédiaires. Cette manière de procéder offre une meilleure visibilité de la solution, et groupe les objets géométriques par type pour fournir un nouveau mécanisme de désignation-sélection.

D'une part l'affichage progressif des calques autorise une explication étape par étape de la construction aux utilisateurs. Précisons qu'avec le mécanisme de calque par type d'objet, la sélection devient non-ambiguë et les objets parasites sont éliminés de la sélection. Dans cette démarche, il y a une correspondance

entre un geste, un type d'objet, et le calque courant.

D'autre part, la sélection multiple est rendue possible en sélectionnant directement un calque, c'est-à-dire tous les objets qu'il contient sans avoir à les choisir directement un par un. Enfin, lorsque l'utilisateur souhaite manipuler toute la scène, il sélectionnera le calque père (ou *racine)* qui englobe tous les autres, c'est-à-dire tous les objets de tous les autres calques. Plutôt que de travailler directement sur un calque, qui agit comme conteneur, il est possible de définir directement un volume de sélection à l'aide de la métaphore bi-manuelle Boîte Englobante (figure 6.11). Chaque objet qui se trouve à l'intérieur de la boîte devient sélectionné.

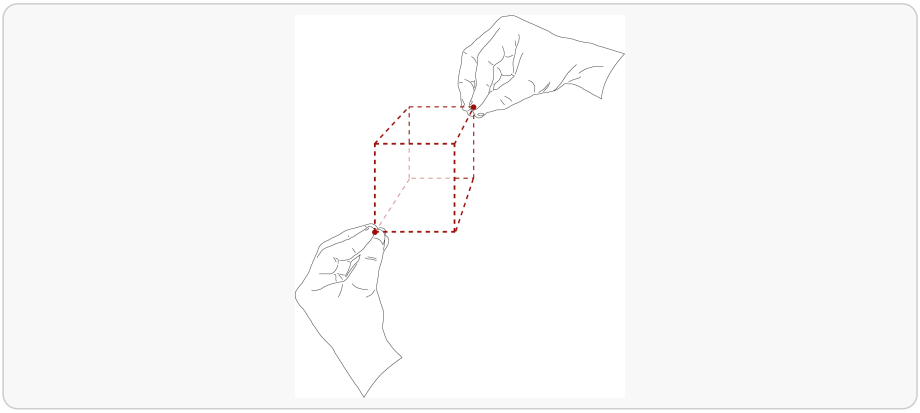


Fig. 6.11: Outil de la boîte englobante.

Dans cette section, nous nous sommes intéressés à l'acte de sélection (désignation puis validation) à l'aide de notre langage de postures. Dans les paragraphes suivants, nous répondrons à la question « comment utiliser ce langage pour construire, manipuler et naviguer ».

6.3.4 Construction bi-manuelle

Dans notre prototype, les deux étapes d'une construction (sélection des objets et sélection d'une primitive de construction), peuvent être réalisées dans n'importe quel ordre. Si une construction est choisie sans objet, le système demande à l'utilisateur de les sélectionner. À l'inverse, si les objets sont déjà sélectionnés, la liste des constructions possibles est proposée.

Nous notons que ces tâches sont souvent réalisées en parallèle. C'est pourquoi nous proposons une utilisation parallèle des mains pour manipuler dans le même temps la scène géométrique et le menu des constructions. Tandis que la main dominante sélectionne des objets dans la scène, la main non dominante interagit avec un menu contextuel dynamique (figure 6.12). L'utilisateur peut alors travailler efficacement sans considérer toutes les possibilités de construction, mais seulement ceux qu'il peut effectivement faire. Ce menu est semi-transparent et les entrées sont sélectionnées *via* la main non dominante. Il apparaît après une

courte période (moins d'une seconde), permettant une sélection aveugle. Par exemple, l'utilisateur choisit 2 points dans l'espace avec sa main dominante. Après une seconde, le menu apparaît et il sélectionne une entrée (par exemple construire la ligne passant par ces points), en poussant sa main non dominante à travers le menu. Ainsi, l'encombrement des menus statiques est réduit. Le menu dynamique s'affiche à hauteur de la main non-dominante ; il revient donc à l'utilisateur de le faire apparaître judicieusement par rapport aux objets d'intérêt.

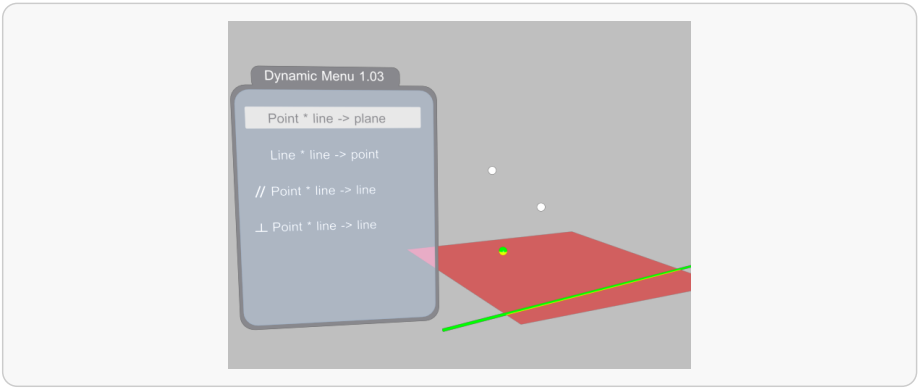


Fig. 6.12: Menu dynamique.

La main non dominante agit dans le même temps, à la fois comme un sélecteur dans le menu, et dans l'espace (sélection d'objet géométrique). Imaginons que l'utilisateur sélectionne un point et une ligne durant la construction. Le système ne peut pas savoir ce que l'utilisateur désire faire : construire une parallèle ou une perpendiculaire. La construction est terminée en choisissant dans le menu dynamique l'icône « parallèle » avec la main non dominante.

Notons que la méthode de construction traditionnelle consiste à demander au système une liste possible des primitives de construction lorsqu'un objet particulier est sélectionné.

6.3.5 Manipulation bi-manuelle

Avec six degrés de liberté (position et orientation), les manipulations sont souvent très difficiles, même en disposant d'indices visuels (ombrage, indices de profondeurs, vues sous un angle différent, etc.). Les manipulations bi-manuelles sont une variante à la manipulation directe qui implique d'utiliser conjointement les deux mains pour une tâche unique. Conformément à la théorie de la chaîne cinématique de Guiard [Gui87], la main non dominante fournit un référentiel à l'utilisateur, ce qui améliore les performances (mouvements de la main dominante plus rapides et plus précis) lors d'une construction géométrique.

Le processus de manipulation peut agir sur deux types d'objets. Premièrement, ceux qui sont directement créés par l'utilisateur (par exemple créer un point libre dans l'espace). Deuxièmement, ceux résultant du processus de

construction (notamment, créer une ligne passant par deux points). Nous proposons deux méthodes pour aider l'utilisateur durant les phases de manipulation et de positionnement.

Grille régulière magnétique

La grille magnétique régulière, illustrée pour la figure 6.13, permet de placer des points, des lignes ou des plans avec précision. Chaque objet géométrique est accroché sur un nœud de la grille. L'espace entre les nœuds peut être ajusté *via* la main non dominante, rendant la grille plus ou moins précise : cette main attrape un axe et s'approche ou s'éloigne du centre de la grille. Avec notre grille magnétique régulière, les problèmes de précision et de profondeur sont notoirement réduits. Bien que le but de cette grille soit de contraindre les déplacements des objets, l'expressivité du modèle géométrique n'est pas réduit puisque le pas de la grille est modifiable.

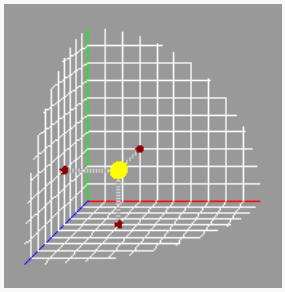


Fig. 6.13: Outil de la grille magnétique régulière.

Dans notre exemple – l'intersection entre un cube et un plan, nous proposons de placer un point libre en premier à l'aide de la grille magnétique régulière. Le plan pourrait également être orienté par positionnement isothétique.

Repère local augmenté

Nous proposons également d'ajouter des références visuelles pour manipuler des objets virtuels. Celles-ci sont placées sur un repère local augmenté, et sont appelées des poignées, pour mettre à l'échelle, tourner et translater l'objet, comme le décrit la figure 6.14.

Ce repère est affiché lorsqu'un objet est manipulé, en étant centré sur lui. Chaque poignée correspond à une modification contrainte possible de l'objet. Nous distinguons les déplacements – rotations et translations – et les mises à l'échelle. Les cônes sont associés à la translation sur un axe, les sphères aux rotations autour d'un axe. Enfin, une mise à l'échelle sera représentée par un cube. Entre deux axes, il existe des plans contenant un petit triangle. Chaque triangle permet de déplacer l'objet dans le plan du triangle uniquement.

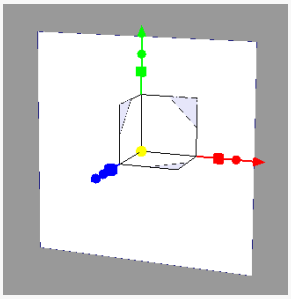


Fig. 6.14: Outil du repère local augmenté.

Dans notre exemple, nous souhaitons expérimenter l'intersection entre un plan et un cube. Ce plan pourrait être déplacé dans une direction, selon sa normale, et a donc un unique degré de liberté. Pour cela, l'utilisateur attrapera un petit cube du repère local augmenté et réalisera des mouvements de translation.

6.3.6 Navigation contrainte

La navigation dans une scène géométrique est peu intuitive. Elle est fortement corrélée avec les déplacements physiques de l'utilisateur, mais ceux-ci sont à proscrire dans des environnements dans lesquels nous travaillons, puisque nous disposons d'un espace assez limité pour se déplacer réellement. Nous proposons d'utiliser une métaphore d'interaction bi manuelle pour explorer la construction géométrique réalisée. En enseignement assisté par ordinateur, cette manière de faire peut s'avérer très efficace et diminuer sensiblement le temps d'apprentissage. En effet, c'est l'enseignant qui choisira le premier les meilleurs angles de vue pour comprendre la construction. Ensuite, les étudiants pourront revister comme ils le souhaitent cette construction, en s'arrêtant aux endroits qu'ils ne comprennent pas bien.

La navigation est basée sur l'utilisation conjointe des mouvements des deux mains, et la métaphore du rayon déformable [SB05]. Notre technique d'interaction par rayon déformable permet de voyager à l'intérieur d'un environnement virtuel. L'utilisateur dessine simplement un chemin $3D$ à l'aide du rayon sur lequel est fixée une caméra : celle-ci offre un seconde point de vue contraint le long d'une courbe. Il s'agit donc de navigation planifiée. Nous pouvons voir, figure 6.15, comment l'utilisation des deux mains permet de manipuler le rayon dans l'espace. Il y a trois modes différents pour l'utiliser : rester sur place, avancer – le rayon s'agrandit – et reculer – le rayon rétrécit. Il existe une zone neutre lorsque les deux mains sont proches – moins de 10 cm. Ensuite, le mode normal est activé lorsque les deux mains sont plus éloignées l'une de l'autre que la distance précédente de neutralisation. Quand cette dernière est fermée – poing fermé, le rayon rétrécit en longueur. À l'inverse lorsque le poing est ouvert, le rayon s'allonge. La vitesse courante est donnée par la distance entre les deux mains, et son orientation courante est celle de la direction du vecteur formée par les deux mains. Précisons que le rayon déformable ne peut tourner sur son propre axe : il est simplement courbé dans l'espace.

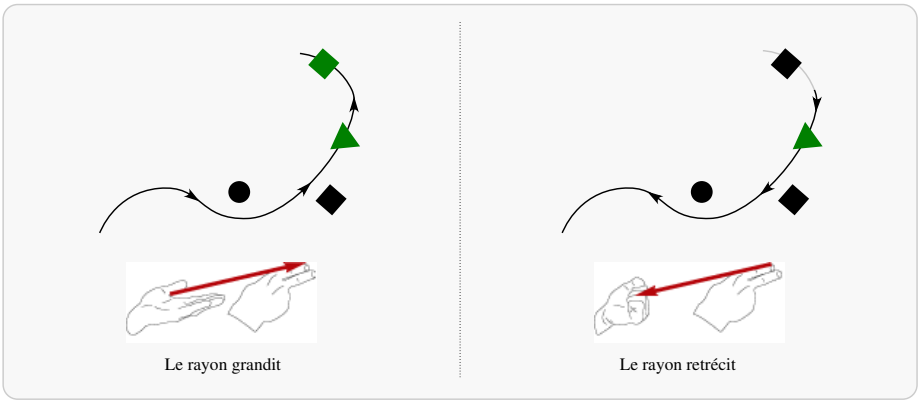


Fig. 6.15: Outil du rayon déformable à deux mains.

Lorsque le rayon a été dessiné dans l'espace, l'utilisateur peut déplacer une caméra virtuelle qui y est accrochée. Il dispose ainsi d'un second point de vue, affiché à côté de la construction. Comme pour la définition du rayon, l'utilisateur est capable de la diriger dans n'importe quelle direction à l'aide de la position relative de la main dominante par rapport à la main non dominante. Les deux mains forment alors un vecteur qui donne à la fois la direction et la vitesse de la caméra sur le rayon.

Dans notre exemple précédent, un enseignant pourrait enregistrer un chemin particulier pour que ses étudiants puissent facilement voir combien de segments (c'est-à-dire quel type de polyèdre) sont générés par l'intersection plan / cube.

6.3.7 Conclusion et perspectives

Une interaction gestuelle contrainte a été présentée dans le domaine des constructions géométriques $3D$. Ses principales fonctionnalités sont :

- la possibilité de contraindre la désignation et la sélection à un type d'objet ;
- un processus de construction contraint par les sélections précédentes ;
- la manipulation contrainte par la représentation des degrés de liberté par poignées ;
- l'emploi d'un rayon déformable pour contraindre l'exploration d'une construction, et un apprentissage optimisé.

Pour le moment, l'application a été conçue pour une utilisation *Workbench*. Cependant, ce dernier n'est ni transportable, ni abordable, ce qui empêche une diffusion et des tests à grande échelle, dans des salles de classe. Nous souhaiterions donc utiliser des gants de données à faible coût comme les gants P5, et un système de suivi des mouvements miniaturisé pour utilisation sur PC standard.

6.4 Autres applications

Nous allons présenter deux applications réalisées par des étudiants dans le cadre de leur projet de master ou de leur stage de DEA. Il s'agit de deux premiers tests grandeur nature pour la vrLib. La première application, STIGMA^{VR}, est la réalisation d'Olivier Thomas dans le cadre de son stage de DEA 2005 – 2006, au sein de notre équipe. La seconde application, SoundShaker^{VR}, est un projet de Master réalisé par Ahlem Ganouchi, Paul Huchelman, Thomas Jund et Lionel Ketterer, également dans notre équipe. Ces applications nous ont permis de saisir avec quelle rapidité et quelle facilité des programmeurs pouvaient développer une application fonctionnelle. Elles nous ont également servi à améliorer la vrLib en testant de nouvelles solutions architecturales plus pertinentes.

6.4.1 STIGMA^{VR}

L'application STIGMA[2] [BBB00] propose de réaliser et visualiser des animations à l'aide d'objets $4D$ et d'un hyper-plan de coupe sur cet objet. Cependant, la manipulation, l'affichage – et donc la compréhension – d'un objet $4D$ en environnement $2D$ sont particulièrement limités. Le portage de cette application en environnement virtuel sous le nom de STIGMA^{VR}, nous permet de tester la pertinence des vues des projections $3D$ de l'animation par objet $4D$.

Modélisation d'objets $4D$ pour l'animation

La modélisation des objets $4D$ repose sur un modèle théorique d'expression des relations topologiques entre les différents constituants d'un objet : les G-cartes [Lie94]. Ce modèle est une extension des cartes combinatoires de [Jac70] et des 2-cartes présentées à la section 6.2. Le plongement est l'opération qui permet d'associer la représentation géométrique d'un objet à sa représentation topologique.

Définition Une carte généralisée de dimension n ($n \geq -1$), ou n-G-carte est $(n+2)$-uplet $G = (B, \alpha_0, ..., \alpha_n)$ tel que :
- B est un ensemble fini non vide dont les éléments sont appelés les brins ;
- $\alpha_0, ..., \alpha_n$ sont des involutions sur B telles que :
  - $\forall i \in 0, ..., n-1$, α_i est sans point fixe : $\forall b \in B$, $b\alpha_i \neq b$,
  - $\forall i \in 0, ..., n-2$, $j \in i+2, ..., n$, $\alpha_i\alpha_j$ est une involution.

Prenons l'exemple de 2 objets topologiquement équivalents : par exemple une boule et un verre. Ils ont la même topologie – même nombre de trous –, cependant leur plongement est différent. Ces deux objets pourraient être représentés par la même G-carte, et disposer pour chacun d'eux, de paramètres géométriques différents.

Les G-cartes sont le constituant de base d'un objet $4D$ au sens de [Bra00] dans son modeleur Stigma. La méthode traditionnelle pour réaliser une animation est de faire intervenir un algorithme d'interpolation entre des points clés. Par exemple, pour animer une voiture, le graphiste indiquera à l'ordinateur "au temps t_0 la voiture se situe là, puis au temps t_1 la voiture est positionnée ici,

[2] pour *Space-Time Interpolation for the Geometric Modeling of Animations*

orientée de telle manière", puis ce sera l'ordinateur qui déterminera la trajectoire la plus plausible. Cependant, ce procédé peut s'avérer coûteux en temps de calculs et de conception lorsque l'animation est complexe.

L'utilisation des objets $4D$ pour l'animation d'objets $3D$ se base sur la notion de coupe d'un objet par un hyperplan. En effet, l'animation d'un objet de dimension n peut être vue comme la succession de coupes d'un objet de dimension $n+1$ à l'aide d'un hyperplan de coupe de dimension n qui se déplace sur une trajectoire prédéfinie. Le modeleur STIGMA est un outil de [Bra00] dédié à la constuction et à la visualisation de tels objets pour l'animation.

Ce modeleur initial conçu sur et pour un environnement de bureau, offre 4 vues $3D$ – XYZ, XYT, XZT, YZT – dans lesquelles est projeté l'objet $4D$, ainsi qu'une projection $2D$, et la séquence d'animation résultante. L'animation est contrôlée par l'utilisateur *via* un simple curseur qui décrit la position de l'hyperplan de coupe avec l'objet $4D$.

Les deux inconvénients majeurs de Stigma sont d'une part la visualisation dans un environnement $2D$, et d'autre part l'interaction de l'objet $4D$ et de l'hyperplan de coupe. En effet, comprendre et manipuler le plongement d'un objet $4D$ sur une station $2D$ n'est pas particulièrement aisée. La réalité virtuelle offre le cadre idéal pour l'animation à base d'objets $4D$.

Architecture

STIGMA^{VR} repose sur un noyau de G-cartes, et DOGME pour la partie objets $4D$, et sur vrLib pour tout ce qui concerne l'interaction et l'interface en environnement virtuel. La figure 6.16 donne une vue schématique de l'architecture de cette application.

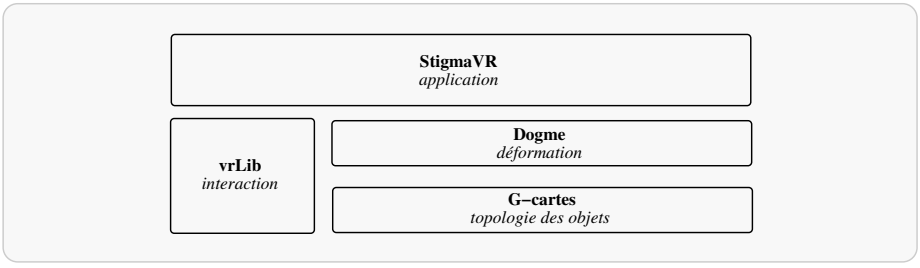


Fig. 6.16: Architecture de l'application STIGMA^{VR}.

Interface graphique et interaction

Les objets $4D$ sont assez lourds en terme de mémoire utilisées. STIGMA^{VR} permet donc de tester le comportement d'une interface construite avec vrLib et de l'interaction lorsque l'application sous-jacente occupe de nombreuses ressources matérielles. La figure 6.17 donne un aperçu de l'interface utilisateur mise en œuvre dans STIGMA^{VR}. Voici les outils dont dispose l'utilisateur pour créer

et visualiser des objets $4D$:

- création d'objets $3D$ et $4D$;
- opérations de base de l'application : réinitialisation de la scène, ouverture et sauvegarde de fichiers, sortie de l'application, etc. ;
- modification des paramètres de visualisation, comme la projection souhaitée pour visualiser l'objet $4D$ dans un espace $3D$, le calcul de la coupe de la scène par un hyperplan, ou le mode de rendu de l'objet ;
- modification de l'hyperplan de coupe qui permet l'affichage de l'animation, et déplacement des objets modélisés.

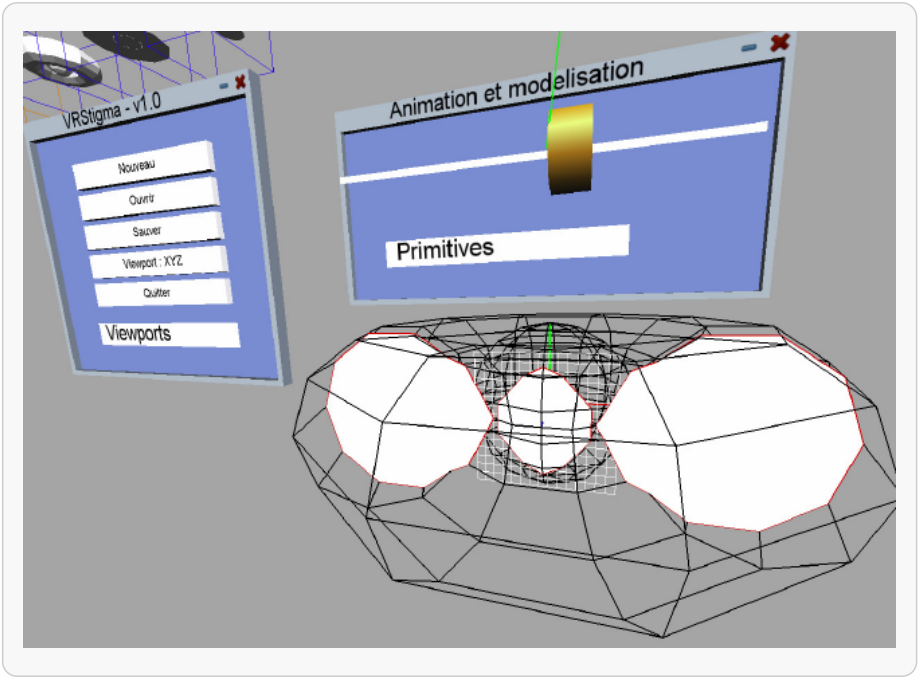


Fig. 6.17: STIGMA^{VR} : visualisation d'une animation $2D$ générée à partir d'un tore et d'un plan de coupe.

Navigation entre les objets $4D$ L'utilisateur peut naviguer dans la scène virtuelle de deux façons : simplement en tournant la tête autour des objets, ce qui limite singulièrement la navigation, ou alors, par attache dans le vide[3]. Avec cette dernière méthode, l'utilisateur ferme le poing dans l'espace, comme s'il souhaitait attraper le vide, puis rapproche ou éloigne sa main selon qu'il souhaite que l'objet avance vers lui, ou au contraire qu'il recule. La navigation se termine lorsque la main est relâchée.

Solution d'affichage d'un objet $4D$ **dans un environnement** $3D$ Dans un modeleur géométrique classique, l'écran est divisé en vues ; généralement, on

[3]a. *grab the air*

en trouve 4 : une vue perspective où l'utilisateur peut tourner autour de l'objet et l'examiner, et 3 vues dédiées avant tout à la modélisation – par exemple insertion ou déplacement de points. Chacune de ces vues correspond à une projection de la scène dans un plan, et dans un modeleur on trouve habituellement les projections sur les XY, YZ et XZ.

Il est impossible d'afficher directement un objet $4D$ dans notre univers qui ne comporte que 3 dimensions. Cependant, il est possible de plonger cet objet dans différentes vues correspondant aux projections dans les espaces XYZ, XYT, XZT et YZT. Dans un environnement $2D$, les vues XY, YZ, XZ et "perspective" sont séparées par des droites. Il nous semblait donc logique de séparer les vues $3D$ par des plans. Les premiers tests que nous avons réalisés ont montré que ce n'est pas la solution idéale, car chaque projection recouvre visuellement les 3 autres.

Pour bien appréhender l'objet $4D$, il est nécessaire de conserver simultanément les 4 vues différentes. Notre idée est de créer un widget Viewport qui contiendra une vue. Ce composant graphique est représenté par un cube au centre duquel se situe une projection de l'objet $4D$; le nom de la projection est affiché au-dessus du widget. La figure 6.18 illustre le placement des 4 vues de projection au dessus de la fenêtre de contrôle de l'application. L'utilisateur dispose donc de 4 points de vue différents pour mieux comprendre et manipuler l'objet $4D$, et les effets des manipulations de l'objet dans les différentes projections.

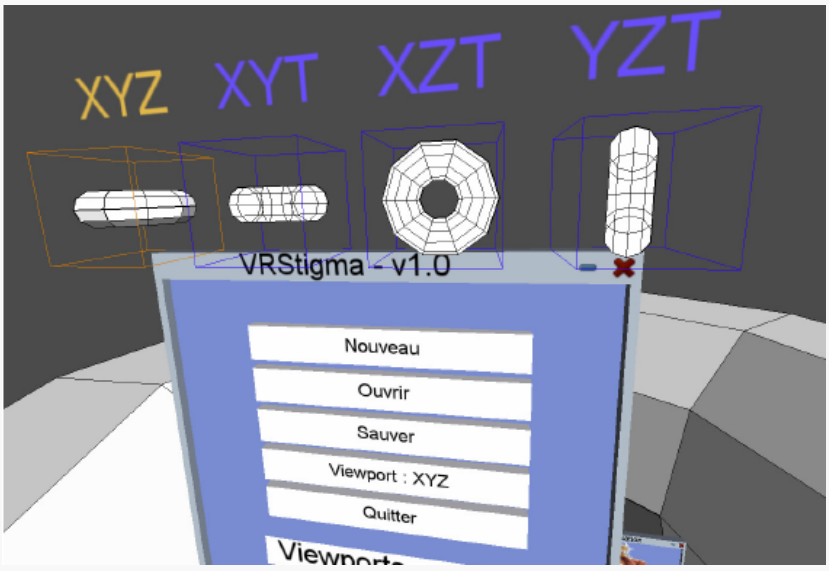


Fig. 6.18: Les 4 vues XYZ, XYT, XZT et YZT de STIGMA^{VR}.

Conclusion

Le portage de l'application Stigma sur un environnement de réalité virtuelle s'avère pertinent pour la création, l'affichage et la manipulation d'une animation $4D$. La vision stéréoscopique permet de projeter un objet $4D$ dans 4 vues différentes et de mieux comprendre l'animation résultante. Les manipulations sont rendues aisées par l'utilisation directe des mains.

Une perspective intéressante est d'utiliser les animations par objets $4D$ dans le cadre d'animation d'une interface graphique. En effet, il pourrait être intéressant d'explorer cette voie dans le cadre du gestionnaire d'effets graphiques de la vrLib.

6.4.2 SoundShaker^{VR}

SoundShaker^{VR} est une application dédiée à la création de sons à l'aide des possibilités d'interaction offertes par un environnement de réalité virtuelle. L'objectif est de pouvoir produire des sons et de les déformer *via* la manipulation d'objets virtuels. L'utilisateur se trouve plongé dans un monde virtuel constitué d'un ensemble d'objets qu'il manipule, déplace et déforme de manière à influer sur les sons générés. L'objectif est de proposer un outil de génération de son en temps réel pour une utilisation en concert, grâce auquel un artiste musical peut se produire sur scène. Ce type de performance existe dans les milieux artistiques, et manipuler des objets géométriques pour produire du son est un concept assez nouveau.

Architecture

L'application SoundShaker^{VR} repose sur la librairie PureData qui est dédiée à la production du son en temps réel à partir de certains paramètres tels que la fréquence, la texture sonore, etc. Chaque paramètre sonore est associé à une déformation d'un objet géométrique, de manière à pouvoir déformer le son en le manipulant visuellement. Les parties interaction et interface graphique sont confiées à la vrLib. La figure 6.19 donne l'architecture mise en œuvre pour SoundShaker^{VR}.

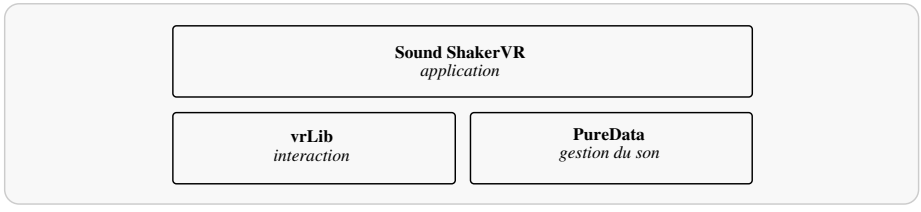


Fig. 6.19: Architecture de l'application SoundShaker^{VR}.

L'utilisateur manipule indirectement deux mécanismes de PureData. D'une part, il peut créer et manipuler des objets sonores. D'autre part, il est également

en mesure de changer le mode de rendu sonore de ces objets, soit en stéréophonie, soit en spatialisation phonique sur un ensemble de haut-parleurs.

Interface graphique et interaction

Les programmeurs ont choisi de créer une interface graphique simple, reposant sur :

- une liste de cubes qui correspondent à des effets sonores différents ;
- un cube permettant de passer du mode *spatialisation* au mode *modification* ;
- des enceintes virtuelles en mode *spatialisation* qui sont placées dans la scène afin d'être capables de connaître le mode d'écoute courant.

Représentation des objets sonores Chaque représentation d'un objet sonore est composée de deux parties distinctes.

La première partie est consacrée au déplacement de l'objet virtuel. Dans le mode *spatialisation*, elle est utilisée pour positionner l'objet sonore par rapport aux quatre enceintes entourant l'utilisateur. En mode *déformation*, elle permet de déplacer la représentation de l'objet sonore de manière à pouvoir convenablement le déformer.

La seconde partie est un ensemble de points à manipuler pour créer une déformation sonore et visuelle sur l'objet. Le détail de cette déformation est donné un peu plus loin.

Interaction manuelle L'utilisation des gants de données et les mécanismes de reconnaissance de postures offerts par la vrLib permettent l'utilisation des mains pour manipuler et déformer les objets de la scène. Pour sélectionner un objet, il suffit de fermer le poing dessus, comme si l'utilisateur souhaitait l'attraper. Pour réduire un objet sonore, l'utilisateur doit rapprocher ses deux mains l'une contre l'autre, paumes ouvertes et pouces dirigés vers le haut. Lorsqu'un objet sonore est réduit, il est mis à l'écart de la scène jusqu'à ce que l'utilisateur le re-sélectionne, auquel cas il reprendra sa dimension et sa position d'origine.

Lorsqu'un objet sonore est sélectionné, il est possible de l'activer et le désactiver en rapprochant les deux mains vers le sol, paumes ouvertes. Visuellement, l'objet passe de la couleur rouge au vert lorsqu'il est activé, et inversement lorsqu'il est désactivé.

Déformation des objets La déformation des objets sonores sert à régler les paramètres du son qu'ils représentent. L'interaction libre, non contrainte, offre d'emblée 6 degrés de liberté – 3 en translation et 3 en rotation. Une première méthode de déformation permet de déplacer une des faces de chaque objet son. Pour réaliser cette opération, l'utilisateur ferme le poing sur une sphère située au centre d'une des faces, et la translation selon la normale de la face sélectionnée.

Une seconde opération, en cours d'implantation, consiste à utiliser simultanément les deux mains pour déformer un objet plus complexe à l'aide du modèle de déformation hélicoïdal Twister de Llamas *et al.* [LKG+03].Chaque face de chaque objet sonore disposera alors de 6 degrés de liberté, au lieu d'un seul, et le retour visuel sera certainement plus pertinent pour le musicien.

Conclusion

Ce projet est centré sur la réalisation d'un générateur virtuel de sons dans l'optique d'une performance musicale en live. Une première version de ce logiciel, basé sur le couplage de deux librairies – vrLib pour l'interface et l'interaction, et PureData pour la gestion du son, a donné des résultats satisfaisants. Pour le moment, l'interaction est essentiellement mono-manuelle et sans modèle de déformation spécifique. Cependant, l'implantation du modèle de déformation bi-manuel Twister offrirait un cadre de travail bien plus large et convivial pour le musicien.

Parmi les pistes à explorer, l'ajout de gestes pourrait permettre de régler le volume global de l'application. Par exemple, avec la main fermée et le pouce levé et dont l'angle formé entre le pouce et le sol correspondrait au volume. On pourrait également imaginer contraindre un objet sonore, comme par exemple le mettre en orbite autour dans autre objet dans la scène, et le coupler à la spatialisation. L'auditeur aurait alors l'impression que le son tourne autour de lui.

6.5 Conclusion

Nous venons de voir 4 applications qui ont été mises en œuvre à l'aide de la librairie d'interaction vrLib. Ces programmes montrent la facilité et la rapidité de conception d'un logiciel dont l'interaction est fonctionnelle. Le coût du développement est sensiblement identique à celui d'une librairie $2D$. MRMaps^{VR} et STIGMA^{VR} ont toutes deux été portées à partie d'une version $2D$. La vrLib permet de transiter d'un environnement de bureau à un environnement immersif de type *Workbench* aisément.

Nous avons cherché à utiliser autant que possible la reconnaissance de postures, *via* les gants de données, de manière à proposer une interaction réellement intuitive et naturelle. Les résultats sont encourageants, et les utilisateurs sont plus à l'aise avec des techniques d'interaction gestuelles qu'avec celles utilisant un périphérique de type *Flystick*. L'introduction de la reconnaissance de gestes dans la vrLib permettra à l'avenir une plus grande palette de métaphores d'interaction.

L'utilisation du matériel comme les capteurs de pression et la surface de contact offerte par l'écran du *Workbench* permet non seulement de contraindre l'interaction pour les objets à dimension fonctionnelle typiquement $2D$, mais aussi une plus grande précision dans les gestes de l'utilisateur. En particulier, les membres supérieurs peuvent se reposer en trouvant appui sur un support physique. Ceci nous paraît indispensable dans le cadre d'une application où

l'interaction doit durer un certain temps.

De nouvelles applications sont en cours de réalisation, notamment sur les modèles de déformation DOGME et Twister. Le cas des contraintes non linéaires de DOGME [BG03] pose des problèmes d'interaction intéressants à résoudre, puisqu'il s'agit de contrôler en même temps bien plus de paramètres que pour les contraintes linéaires.

Conclusion et perspectives

Nous concluons par les caractéristiques du travail présenté dans ce mémoire, ses limitations, et les perspectives de recherche qu'il ouvre.

Bilan

L'objectif de ce mémoire est de présenter des solutions permettant d'améliorer l'interaction en réalité virtuelle.

État de l'art Notre approche de l'état de l'art sur l'interaction en réalité virtuelle offre une vision globale des outils qui existent dans ce domaine. Nous revisitons l'état de l'art sur l'interaction afin de le compléter par les techniques récentes, en y ajoutant notamment un nouveau chapitre sur l'entrée de symboles. L'originalité de ce travail est de l'inscrire, de manière claire et précise, dans le triptyque de l'interaction : le matériel, le logiciel et les sens de l'homme que l'on cherche à stimuler. Une partie de nos écrits a été incluse dans le livre *Le traité de la réalité virtuelle* de Fuchs *et al.* [FMBC06], dans le chapitre *Les techniques d'interaction pour les primitives comportementales virtuelles.*

Librairie vrLib La librairie vrLib est une librairie permettant l'écriture d'applications de réalité virtuelle. Elle permet soit de porter une application $2D$ en $3D$, soit d'écrire un programme complet pour un environnement immersif. Sa structure a été conçue pour être à la fois souple, robuste et simple à utiliser. Différents mécanismes de gestion des contraintes, et des effets graphiques sur les objets autorisent une grande liberté d'interaction. Les outils sont très facilement extensibles selon les besoins du programmeur.

Avant la vrLib, il n'existait à notre connaissance aucune librairie d'interaction qui mette à disposition du programmeur des outils qui permettent d'écrire aussi rapidement et simplement une application pleinement fonctionnelle.

Techniques d'interaction en réalité virtuelle Notre travail de recherche sur les nombreuses techniques d'interaction, consacrées à la réalité virtuelle, nous a amené à reconsidérer l'interaction plus globalement que ce qui avait été réalisé jusqu'à présent. Nous commençons par considérer toute la chaîne d'interaction : homme $\leftrightarrow$ périphérique d'interaction $\leftrightarrow$ périphérique d'affichage $\leftrightarrow$ entité virtuelle, avec laquelle il interagit, alors que la plupart des auteurs se focalisent sur un des maillons de cette chaîne. Par exemple, malgré de nombreuses

tentatives, il est admis qu'utiliser un menu $2D$ dans un environnement $3D$ n'est pas très efficace. Or, c'est peut-être l'outil de travail qui ne convient pas au menu, et non le menu lui-même.

Nous avons proposé un ensemble de règles et de lois de conception, à respecter lors de l'élaboration des objets de l'interface, et des outils d'interaction qui agiront dessus. L'implication la plus intéressante de ce travail est l'inclusion dans la librairie vrLib d'une méthode dite d'interaction adaptative, grâce à laquelle un outil d'interaction peut s'adapter automatiquement aux objets de l'interface. Il s'agit donc clairement d'une amélioration notable de l'interaction en réalité virtuelle. En particulier, nous avons montré que les widgets $2D$ utilisés dans un environnement $3D$ ont leur place, et, qui plus est, s'avèrent nécessaires pour de nombreuses tâches.

Dans notre travail sur l'interaction gestuelle, nous nous sommes appliqués à considérer l'ensemble des problèmes liés à la reconnaissance des postures et des gestes, depuis le suivi des mains dans l'espace, jusqu'à la reconnaissance des gestes. L'intérêt que nous avons porté à stabiliser le suivi de mouvements a permis de coupler les algorithmes ICP et le filtre de Kalman. Nous avons obtenu des résultats satisfaisants, les mouvements étant nettement plus cohérents avec cette correction.

Il a ensuite été possible d'utiliser les gants de données pour reconnaître des postures. Nous avons testé deux méthodes différentes : la première, très simple et repose sur un mécanisme de comparaison d'angles de flexion, tandis que la seconde utilise un réseau de neurones. Les taux de reconnaissance sont très corrects pour cette dernière méthode. Notre apport sur l'interaction gestuelle a également consisté à définir une méthode de reconnaissance de gestes à l'aide d'une grammaire simple, et de la technique de la boussole $3D$.

Applications Deux applications principales nous ont servi à tester en conditions réelles le fonctionnement de la vrLib pour la conception d'une application complexe et nécessitant une interaction simple et performante. La première, MRMaps^{VR}, permet d'éditer des surfaces de subdivisions multi-résolution. Nous avons établi que vrLib était un excellent outil pour porter une application $2D$ en environnement $3D$, tout en étant suffisamment souple pour étendre convenablement l'interaction. La seconde application, Cad^{VR}, a pour objectif de poser des contraintes géométriques en $3D$. MRMaps^{VR} et Cad^{VR} se manipulent toutes les deux entièrement avec les mains, ce qui démontre que l'interaction gestuelle que nous avons imaginée fonctionne correctement.

Plusieurs autres applications ont été écrites, ou sont en cours d'écriture avec la vrLib. Cette librairie semble très bien convenir aux programmeurs et aux utilisateurs, et de nouveaux projets basés dessus sont en cours de mise au point.

Limitations et perspectives

Nous distinguons plusieurs limitations dans notre travail.

limites

Notre état de l'art a pour principale limitation de ne pas être exhaustif, et il nous semble d'ailleurs improbable de vouloir dresser une classification complète de l'ensemble du matériel et des techniques d'interaction disponibles aujourd'hui. En effet, le nombre de périphériques et de métaphores logicielles augmente sans cesse. En introduction à ce manuscrit, nous soulevions d'ailleurs la nécessité d'arriver à uniformiser et standardiser les aspects matériels et logiciels pour l'interaction. Le domaine est vaste, et le fait que la recherche se tourne de plus en plus vers les autres sens que la vue n'enlève rien à la difficulté des deux besoins dont nous venons de discuter, bien au contraire.

Les limitations sur la librairie vrLib sont les suivantes. En premier lieu, bien qu'une première version ait été implantée, il manque toujours un gestionnaire convenable de description d'interface – par exemple en XML –. Il serait judicieux de pouvoir paramétrer l'interaction et l'interface à l'aide d'un langage de scripts. Nous voyons une seconde limitation importante à notre librairie : elle propose peu de métaphores pleinement $3D$. L'essentiel de l'interaction est assuré par les postures et les mains ; en outre, une méthode de reconnaissance de postures statistique fait défaut, car elle constitue selon nous le maillon manquant à l'implantation de notre méthode de reconnaissance de gestes. Enfin, l'accès aux événements système ne convient pas toujours, et peut apporter un surcoût de développement. En fait, la méthode actuelle de gestion de ces événements autorise le programmeur qui manque de rigueur à alourdir sensiblement son propre code.

perspectives

Nous nous appliquons à améliorer la théorie de l'interaction dirigée par la dimension. La direction de recherche que nous envisageons devrait nous conduire vers la loi de Fitts [FP64, GB04b]. En effet, celle-ci traite des problèmes de taille des widgets – plus globalement des objets –, de leur placement, de la taille des outils d'interaction, de la distance entre l'outil et l'objet. Si l'on regarde attentivement les travaux sur la loi de Fitts, on peut remarquer qu'ils ne considèrent que le volume de la cible, et non le volume de l'outil de sélection. Il y a, selon nous, deux problèmes importants :

- on ne considère pas du tout le volume de sélection, ce qui revient à sélectionner un volume par un volume ;
- on ne considère pas explicitement la dimension fonctionnelle d'interaction courante qui pourrait paramétrer la loi de Fitts, en fonction de la dimension considérée. Pour l'instant, il existe une loi par dimension.

Nous pensons qu'il pourrait être intéressant de s'attarder à étudier une loi de Fitts généralisée paramétrée par la dimension fonctionnelle. Cela reviendrait à mettre en relation :

- le temps de réalisation de la tâche ;
- la dimension de la cible (ex : widget) ;
- la dimension de l'outil d'interaction ;
- l'hyper-taille de la cible dans la dimension considérée ;
- l'hyper-taille du volume de sélection dans la dimension considérée ;

– la distance entre le volume de sélection et la cible.

Le matériel employé pour suivre les mouvements de l'utilisateur est très encombrant et onéreux. Ces problèmes conduisent à la nécessité de concevoir une brique logicielle de suivi de mouvements par webcam à faible coût, pour introduire la réalité virtuelle dans les environnements de bureau. Il s'agit d'une solution peu onéreuse pour migrer en douceur vers des environnements de réalité virtuelle de taille réduite. Notre objectif serait de pouvoir apporter des outils de réalité virtuelle utilisables sur station aux enseignements de mathématiques par exemple, afin de les aider à montrer à leurs étudiants certaines configurations géométriques. L'utilisateur disposerait ainsi d'un environnement qui conserve les avantages de la $2D$ tout en ajoutant ceux de la $3D$.

En environnement $2D$, un programmeur peut d'ordinaire utiliser un concepteur graphique d'interface pour construire l'interface de ses programmes. Pour cela, il lui suffit de construire cette dernière à l'aide de la souris. vrDesigner s'inscrit dans une perspective à long terme. Il s'agit d'un studio de programmation graphique à la manière de ceux disponibles en environnement $2D$, avec lequel le développeur placerait visuellement ses widgets, et sélectionnerait les outils d'interaction adéquats pour chaque widget sélectionné.

La dernière perspective que nous voyons à notre travail est un bureau de travail $3D$. Un tel outil de travail permettrait de manipuler visuellement l'ensemble des applications, comme par exemple lancer un ou plusieurs logiciels simultanément sans que l'un d'eux soit exclusif par rapport aux autres. Il s'agit d'une idée qui se situe à l'intersection de nos différentes préoccupations, et envisageable sur le long terme.

Bibliographie

[3DW] 3DWM. http ://www.cs.chalmers.se/ẽlm/projects/3dwm/. Web site.

[AB92] Steve Aukstakalnis and David Blatner. *Silicon Mirage : The Art and Science of Virtual Reality*. Peach Pit Press, 1992.

[AB06] Anand Agarawala and Ravin Balakrishnan. Keepin' it real : Pushing the desktop metaphor with physics, piles and the pen. In *Proceedings of CHI'06*, 2006.

[ABW93] Kevin W. Arthur, Kellogg S. Booth, and Colin Ware. Evaluating 3d task performance for fish tank virtual worlds. *Information Systems*, 11(3) :239–265, 1993.

[Arn02] Laura Lynn Arns. *A New Taxonomy for Locomotion in Virtual Environments*. PhD thesis, Iowa State University, 2002.

[ASMR02] Dzmitry Aliaskseyeu, Sriram Subramanian, Jean-Bernard Martens, and Matthias Rauterberg. Interaction techniques for navigation through and manipulation of 2d and 3d data. In *Proceedings of Eight Eurographics Workshop on Virtual Environments*, pages 179–188, 2002.

[BAHO92] Jeff Butterworth, Davidson Andrew, Stephen Hench, and Marc Olano. 3dm : A three dimensional modeler using a head-mounted display. In *Symposium on Interactive 3D Graphics*, pages 135–138, 1992.

[Bar84] A.H. Barr. Global and local deformations of solid primitives. In *Proceedings of SIGGRAPH '84*, pages 21–31, 1984.

[Bau91] Yves Baulac. Cabri-géométre, a tool for computer aided geometry. In *Wheels for the mind*, pages 30–34, 1991.

[BB91] P. Borrel and D. Bechmann. Deformation of n-dimensional objects. *International Journal of Computational Geometry & Applications (Special Issue : Solid Modeling II)*, 1(4) :427–453, 1991.

[BBB00] S. Brandel, D. Bechmann, and Y. Bertrand. The thickening : an operation for animation. *The Journal of Visualization and Computer Animation*, 1(5) :261–277, 2000.

[BCSH+92] 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*, volume 26, pages 183–188, 1992.

[BDA99] Patrick Bourdot, Martin Dromigny, and Laurent Arnal. Virtual navigation fully controlled by the head tracking. In *Proceedings of International Scientific Workshop on Virtual Reality and Prototyping - Laval'99*, 1999.

[BDHN99] Doug A. Bowman, Elizabeth T. Davis, Larry F. Hodges, and Badre Albert N. Maintaining spatial orientation during travel in an immersive virtual environment. In *Presence : Teleoperators and Virtual Environments*, volume 8(6), pages 618–631, 1999.

[BFB02] Davide Brunelli, Elisabetta Farella, and Maria Elena Bonfigli. Untethered interaction for immersive virtual environments through handheld devices. In *Eurographics Italian Chapter*, 2002.

[BG03] Dominique Bechmann and Dominique Gerber. Arbitrary shaped deformations with dogme. *The Visual Computer*, 2003.

[BH95] W. Barfield and C. Hendrix. The effect of update rate on the sense of presence within virtual environments. *Virtual Reality : The Journal of the Virtual Reality Society*, pages 3–16, 1995.

[BH97] Doug A. Bowman and Larry F. Hodges. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *Proceedings of ACM Symposium on Interactive 3D Graphics*, pages 35–38, 182, 1997.

[BH99] Doug A. Bowman and Larry F. Hodges. Formalizing the design, evaluation, and application of interaction techniques for immersive virtual environments. *Journal of Visual Languages and Computing*, 10(1) :37–53, 1999.

[Bil01] Mark Billighurst. Crossing the chasm. In *Proceedings of International Conference on Augmented Tele-Existence 2001*, 2001.

[BKH97] Doug A. Bowman, David Koller, and Larry F. Hodges. Travel in immersive virtual environments : An evaluation of viewpoint motion control techniques. *Proceedings of IEEE Virtual Reality Annual International Symposium*, 7 :45–52, 1997.

[BKH98] Doug A. Bowman, David Koller, and Larry F. Hodges. A methodology for the evaluation of travel techniques for immersive virtual environments. *Virtual Reality : Research, Development, and Applications*, 3(2) :120–131, 1998.

[BKLJP01] Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola Jr., and Ivan Poupyrev. An introduction to 3-d user interface design. In *Presence*, volume 10(1), 2001.

[BKLP05] Doug Bowman, E. Kruijff, J. LaViola, and I. Poupyrev. *3D User Interfaces : Theory and Pratice*. Addison-Wesley, 2005.

[BLC01] Doug A. Bowman, Vinh Q. Ly, and Joshua M. Campbell. Pinch keyboard : Natural text input for immersive virtual environments. Technical report, Computer Science, Virginia Tech, 2001.

[BM92] P. Besl and N. McKay. A method for registration of 3-d shapes. In *Transaction of PAMI*, volume 14(2), 1992.

[BMH98] D. Brogan, R. Metoyer, and J. Hodgins. Dynamically simulated characters in virtual environments. In *IEEE Computer Graphics and Application*, pages 58–69, 1998.

[Bol80] Richard A. Bolt. Put-that-there : voice and gesture at the graphics interface. In *7th annual conference on Computer graphics and interactive techniques*, 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.

[Bra00] Sylvain Brandel. *Conception et implantation d'un modeleur 4-D pour l'animation*. PhD thesis, Université Louis Pasteur, Strasbourg, 2000.

[BW01] Doug A. Bowman and Chadwick A. Wingrave. Design and evaluation of menu systems for immersive virtual environments. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 149–156, 2001.

[BWHA98] Doug A. Bowman, Jean Wineman, Larry F. Hodges, and Don Allison. Designing animal habitats within an immersive ve. IEEE Computer Graphics and Applications, 18(5), 9-13, 1998.

[CC78] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6) :350–355, 1978.

[CD00] Ashley Gadd A. Chou, T. and Knott D. A vision based approach to data glove calibration. In *Hand-Eye :Proceedings of Human Interface Technologies*, pages 47–54, 2000.

[CGL00] Sabine Coquillart, Jérôme Grosjean, and Anatole Lécuyer. Interaction et systèmes haptiques. PBVE 2000 (Environnements Virtuels à Base de Projection sur Grands Ecrans), Séminaire INRIA, 2000, 2000.

[Cha99] Stéphane Channac. *Conception et mise en oeuvre d'un système déclaratif de géométrie dynamique*. PhD thesis, Université Joseph Fourier - Grenoble 1, 1999.

[CM92] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Image and Vision Computing*, pages 145–155, 1992.

[CNSD93] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. Surround-screen projection-based virtual reality : The design and implementation of the cave. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 135–142, 1993.

[Com95] Palm Computing. Suddenly newton understands everything you write. *Pen Computing Magazine*, page 9, 1995.

[DAA98] Rudolph P. Darken, Terry Allard, and Lisa B. Achille. Spatial orientation and wayfinding in large-scale virtual spaces. In *Presence*, volume 7(2), pages 101–107, 1998.

[DAA99] Rudolph P. Darken, Terry Allard, and Lisa B. Achille. Spatial orientation and wayfinding in large-scale virtual spaces ii : Guest editors' introduction. *Presence*, 8(6) :iii–vi, 1999.

[DAFS01] R. De Amicis, M. Fiorentino, and A. Stork. Parametric interaction for cad application in virtual reality environment. In *XII International Conference, International Conference on Design Tools and Methods in Industrial Engineering*, pages 43–52, 2001.

[DC99] Rudolph P. Darken and Helsin Cevik. Map usage in virtual environments : Orientation issues. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 133–140, 1999.

[DC00] James Davis and Xing Chen. Lumipoint : Multi-user laser-based interaction on large tiled displays. *Displays*, 23(5), 2000.

[DCC97] Rudolph P. Darken, William R. Cockayne, and David Carmein. The omni-directional treadmill : A locomotion device for virtual worlds. In *ACM Symposium on User Interface Software and Technology*, pages 213–221, 1997.

[DL94] Rudy Darken and The U.S. Naval Research Laboratory. Hands-off interaction with menus in virtual spaces. In Scott S. Fisher, John O. Merritt, and Mark T. Bolas, editors, *Proceedings of SPIE : Visual Data Interpretation*, volume 2177, pages 365–371, 1994.

[DLG90] N. Dyn, D. Levin, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. Graph.*, 9(2) :160–169, 1990.

[DPC99] Cédric Dumas, Patricia Plénacoste, and Christophe Chaillou. Définition d'un modèle d'interaction pour une interface de travail tridimensionnelle à partir d'expérimentations. In *Proceedings of IHM'99*, 1999.

[DS78] D. Doo and M. Sabin. Analysis of the behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6) :356–360, 1978.

[DW00] M. Droske and G. Wesche. Conceptual free-form styling on the responsive workbench. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology*, 2000.

[EGJ93] R. A. Earnshaw, M. A. Gigante, and H. Jones. *Virtual Reality Systems*. New York Academic Press, 1993.

[EH97] John Edwards and Chris Hand. Maps : Movement and planning support for navigation in an immersive vrml browser. In *VRML'97*, 1997.

[Ell91] S.R. Ellis. Nature and origins of virtual environments : A bibliographical essay. *Computing Systems in Engineering*, 2(4) :321–347, 1991.

[ENK97] T. Todd Elvins, David R. Nadeau, and David Kirsch. Worldlets - 3d thumbnails for wayfinding in virtual environments. *ACM Symposium on User Interface Software and Technology*, pages 21–30, 1997.

[Fab06] Arnaud Fabre. *Contraintes géométriques en dimension 3*. PhD thesis, Université Louis Pasteur, Strasbourg, 2006.

[FH98] S. Sidney Fels and Geoffrey E. Hinton. Glove-talk ii : A neural network interface which maps gestures to parallel formant speech synthesizer controls. *IEEE Transactions on Neural Networks*, 9(1) :205–212, 1998.

[FHZ96] Andrew Forsberg, Kenneth Herndon, and Robert Zeleznik. Aperture based selection for immersive virtual environments. In *Proceedings of ACM Symposium on User Interface Software and Technology*, pages 95–96, 1996.

[Fit93] George W. Fitzmaurice. Situated information spaces and spatially aware palmtop computers. *Communications of the ACM*, 36(7) :39–49, 1993.

[FMBC06] Philippe Fuchs, Guillaume Moreau, Jean-Marie Burkhardt, and Sabine Coquillart. *Le traité de la réalité virtuelle - Volume 2*. Presses de l'Ecole des Mines de Paris, 2006.

[FMS04] Arnaud Fabre, Pascal Mathis, and Pascal Schreck. 3d geometric constructions in virtual reality. In *IEEE Virtual Reality International Conference Laval Virtual*, 2004.

[FP64] P. Fitts and J. Peterson. A new approach to linear filtering and prediction problems. *Information Capacity of Discrete Motor Responses. Journal of Experimental Psychology*, 67 :103–112, 1964.

[Fre] FreeVR. http ://www.freevr.org. Web site.

[FSS97] Chris Faisstnauer, Dieter Schmalstieg, and Zsolt Szalavári. Device-independent navigation and interaction in virtual environments. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, 1997.

[FSS98] Chris Faisstnauer, Dieter Schmalstieg, and Zsolt Szalavári. Computer-assisted selection of 3d interaction and navigation metaphors. In *Proceedings of Workshop on Computer Graphics*, 1998.

[Fuc03] Philippe Fuchs. *Le traité de la réalité virtuelle*. Ecole des mines de Paris, 2003.

[FVDFH90] J. Foley, A. Van Dam, S. Feiner, and J. Hugues. *Computer graphics Principles and Practice, 2e edition*. Addison-Wesley, 1990.

[FVdSH98] M. Fischer, P. Van de Smagt, and G. Hirzinger. Learning techniques in a dataglove based telemanipulation system for the dlr hand. *Transactions of the IEEE International Conference on Robotics and Automation*, pages 1603–1608, 1998.

[Gal95] Tinsley A. Galyean. Guided navigation of virtual environments. In *Proceedings of Symposium on Interactive 3D Graphics*, pages 103–104, 1995.

[GB04a] Dominique Gerber and Dominique Bechmann. Design and evaluation of the ring menu in virtual environments. In *Proceedings of Immersive Projection Technology*, 2004.

[GB04b] T. Grossman and R. Balakrishnan. Pointing at trivariate targets in 3d environments. In *Proceedings of SIGCHI04*, 2004.

[GB05] Dominique Gerber and Dominique Bechmann. The spin menu : A menu system for virtual environments. In *Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*, pages 271–272, 2005.

[GC01] Jérôme Grosjean and Sabine Coquillart. Command & control cube : a shortcut paradigm for virtual environments. In *Proceedings of IPT-EGVE'01*, 2001.

[GC03] Ludovic Garreau and Nadine Couture. Study of tangible user interface for handling tridimensionnal objects. In *Prooceeding of Workshop on Real World User Interfaces*, pages 64–68, 2003.

[Geo] Geospacw. http ://www2.cnam.fr/creem/. Web Site.

[GL04] Mark Green and Joe Lo. The grappl 3d interaction technique library. In *Proceedings of the ACM symposium on Virtual reality software and technology VRST '04*, pages 16–23, 2004.

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

[Gui87] Y. Guiard. Asymmetric division of labor in human skilled bimanual action : The kinematic chain as a model. *Journal of Motor Behavior*, 19 :486–517, 1987.

[Hac03] Martin Hachet. *Interaction avec des environnements virtuels affichés au moyen d'interfaces de visualisation collective*. PhD thesis, Université Bordeaux I, 2003.

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

[HBCN02] Patrick L. Hartling, Allen D. Bierbaum, and Carolina Cruz-Neira. Tweek : Merging 2d and 3d interaction in immersive environments. In *6th World Multiconference on Systemics, Cybernetics, and Informatics*, 2002.

[HL93] Richard Holloway and Anselmo Lastra. Virtual environments : A survey of the technology. Technical Report TR93-033, University of North Carolina at Chapel Hill, 1993.

[Hol02] John M. Hollerbach. *Locomotion Interfaces in : Handbook of Virtual Environments Technology*. Lawrence Erlbaum Associates, 2002.

[HPGK94] K. Hinckley, R. Pausch, J. Goble, and N. Kassell. Passive real-world interface props for neurosurgical visualization. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, pages 452–458, 1994.

[HPRB96] Reid Harmon, Walter Patterson, William Ribarsky, and Jay Bolter. The virtual annotation system. Technical Report 96-01, Proceedings of IEEE Virtual Reality Annual International Symposium, 1996.

[HTP+97] Ken Hinckley, Joe Tullio, Randy F. Pausch, Dennis Proffitt, and Neal F. Kassell. Usability analysis of 3d rotation techniques. In *Proceedings of ACM Symposium on User Interface Software and Technology*, pages 1–10, 1997.

[HvDG94] Kenneth P. Herndorn, Andries van Dam, and Michael Gleicher. Workshop on the challenges of 3d interaction. In *Issue of SIG CHI'94 Bulletin*, volume 26-4, pages 365–371, 1994.

[IKMT98] Takeo Igarashi, Rieko Kadobayashi, Kenji Mase, and Hidehiko Tanaka. Path drawing for 3d walkthrough. In *ACM Symposium on User Interface Software and Technology*, pages 173–174, 1998.

[IMT99] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy : A sketching interface for 3d freeform design. In *Proceedings of SIGGRAPH '99 Conference*, 1999.

[IYUM04] Hiroo Iwata, Hiroaki Yano, Takahiro Uemura, and Tetsuro Moriya. Food simulator : A haptic interface for biting. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 51–57, 2004.

[Jac] Nicholas Jackiw. The geometer's sketchpad. key curriculum, 1991-1995. Berkeley.

[Jac70] A. Jacques. Constellation et graphes topologiques. In *Combinatorial Theory and Applications*, pages 657–673, 1970.

[JE92] Richard H. Jacoby and Stephen R. Ellis. Using virtual menus in a virtual environment. In Joanna R. Alexander, editor, *Proceedings of SPIE : Visual Data Interpretation*, volume 1668, pages 39–48, 1992.

[Kal60] R. Kalman. A new approach to linear filtering and prediction problems. *Transaction of the ASME – Journal of Basic Engineering*, pages 35–45, 1960.

[Kau02] Hannes Kaufmann. Construct3d : An augmented reality application for mathematics and geometry education. In *In Proceedings of ACM Multimedia Conference 2002*, 2002.

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

[KCB06] P. Kraemer, D. Cazier, and D. Bechmann. Extension multirésolution des cartes combinatoires : application à la représentation des surfaces de subdivision multirésolution. In *Proceedings of AFIG'06*, 2006.

[KHK99] Yoshifumi Kitamura, Tomohiko Higashi, and Fumio Kishino. Virtual chopsticks : Object manipulation using multiple exact interactions. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 198–204, 1999.

[KIK01] Yoshifumi Kitamura, Yuichi Itoh, and Fumio Kishino. Real-time 3d interaction with activecube. In *Proceedings of CHI'01*, pages 355–356, 2001.

[Kor99] Ulrich Kortenkamp. *Foundations of Dynamic Geometry*. PhD thesis, Swiss Federal Institue of Technology, Zurich, 1999.

[KP00] Michal Koutek and Frits Post. Dynamic object manipulation in a virtual workbench environment. In *Proceedings of ASCI'00*, pages 364–371, 2000.

[KP01] Michal Koutek and Frtis H. Post. Spring-based manipulation tools for virtual environments. In *Proceedings of Immersive Projection Technology and Virtual Environments*, pages 61–70, 2001.

[KP02] Michal Koutek and Frits Post. Data visualization in virtual reality, 2002.

[KP05] Ernst Kruijff and Aeldrik Pander. Experiences of using shockwaves for haptic sensations. In *IEEE VR 2005 Workshop*, 2005.

[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, Jeoren van Hees, Frit H. Post, and A.F. Bakker. Virtual spring manipulators for particle steering in molecular dynamics on the responsive workbench. In *Proceedings of the workshop on Virtual environments 2002*, pages 53–63, 2002.

[KZK04] F. Kahlesz, G. Zachmann, and R. Klein. Visual-fidelity dataglove calibration. In *Proceedings of Computer Graphics International*, pages 403–410, 2004.

[LBE04] A. Lécuyer, J.M. Burkhardt, and L. Etienne. Feeling bumps and holes without a haptic interface : the perception of pseudo-haptic textures. In *Proceedings of ACM Conference in Human Factors in Computing Systems (ACM SIGCHI)*, 2004.

[LCK+00] A. Lécuyer, S. Coquillart, S. Kheddar, P. Richard, and P. Coiffet. Pseudo-haptic feedback : Can isometric input devices simulate force feedback ? In *Proceedings of IEEE Virtual Reality Annual International Symposium*, 2000.

[LDP+02] James Jeng-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.

[LG93] Liang and Green. Jdcad : a highly interactive 3d modeling system. In *Computer Graphics (SIGGRAPH'93 Proceedings)*, volume 18(4), pages 499–506, 1993.

[Lie94] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. In *International Journal of*

Computational Geometry and Applications, pages 373–389, 1994.

[Lin03] Robert W. Linderman. Virtual contact : The continuum from purely visual to purely physical. In *Proceedings of the 47th Annual Meeting of the Human Factors and Ergonomics Society (HFES)*, 2003.

[LJ00] Joseph J. LaViola Jr. A discussion of cybersickness in virtual environments. *ACM SIGCHI Bulletin*, 32(1) :47–56, 2000.

[LJAFKZ01] Joseph J. LaViola Jr., Daniel Acevedo Feliz, Daniel F. Keefe, and Robert C. Zeleznik. Hands-free multi-scale navigation in virtual environments. In *Proceedings of Symposium on Interactive 3D Graphics*, pages 9–15, 2001.

[LKG+03] Ignacio Llamas, Byungmoon Kim, Joshua Gargus, Jarek Rossignac, and Chris D. Shaw. Twister : a space-warp operator for the two-handed editing of 3d shapes. In *Proceedings of ACM SIGGRAPH 2003*, volume 22(3), pages 663–668, 2003.

[Loo87] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.

[LSH99] Robert W. Lindeman, John L. Sibert, and James K. Hahn. Hand-held windows : Towards effective 2d interaction in immersive virtual environments. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 205–212, 1999.

[LST01] Robert W. Lindeman, John L. Sibert, and James N. Templeman. The effect of 3d widget representation and simulated surface constraints on interaction in virtual environments. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 141–148, 2001.

[MA98] Jennifer Mankoff and Gregory D. Abowd. Cirrin : A word-level unistroke keyboard for pen input. In *ACM Symposium on User Interface Software and Technology*, pages 213–214, 1998.

[MBJS97] Mark R. Mine, Frederick P. Brooks Jr., and Carlo H. Sequin. Moving objects in space : Exploiting proprioception in virtual-environment interaction. *Computer Graphics*, 31(Annual Conference Series) :19–26, 1997.

[MBN+02] Brad A. Myers, Rishi Bhatnagar, Jeffrey Nichols, Choon Hong Peck, Dave Kong, Robert Miller, and A. Chris Long. Interacting at a distance : Measuring the performance of laser pointers and other devices. In *Proceedings of CHI'02*, volume 4(1), pages 33–40, 2002.

[MCK03] Arnold Müller, Stefan Conrad, and Ernst Kruijff. Multifaceted interaction with a virtual engineering environment using a scenegraph-oriented approach. In *Proceeding of WSCG'2003*, 2003.

[MCR90] Jock D. Mackinlay, Stuart K. Card, and George G. Robertson. Rapid controlled movement through a virtual 3d workspace.

In *Computer Graphics (SIGGRAPH'90 Proceedings)*, pages 171–176, 1990.

[Mec] Roland Mechling. http ://www.mechling.de. Web site.

[MFS05] C. Matabosch, D. Fofi, and J. Salvi. Reconstruction et recalage de surfaces en mouvement par projection laser multilignes. In *9es Journées des Jeunes Chercheurs en Vision par Ordinateur (ORASIS'2005)*, 2005.

[Min95] Mark R. Mine. Virtual environment interaction techniques. Technical Report TR95-018, Departement of Computer Science, University of North California, Chapel Hill, NC 27599-3175, 1995.

[Min96] Mark R. Mine. Working in a virtual world : Interaction techniques used in the chapel hill immersive modeling program. Technical Report TR96-029, Departement of Computer Science, University of North California, Chapel Hill, NC 27599-3175, 1996.

[MM95] Daniel J. Mapes and Michael J. Moshell. A two-handed interface for object manipulation in virtual environments. *Presence*, 4(4) :403–416, 1995.

[MRWBJ03] 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.

[MS94] Thomas H. Massie and J.K. Salisbury. The phatom haptic interface : A device for probing virtual objects. In *Proceedings of the ASME Dynamic Systems and Control Division*, volume 55-1, pages 295–300, 1994.

[MS01] Tim Marsh and Shamus P. Smith. Guiding user navigation in virtual environments using awareness of virtual off-screen space, 2001.

[MT91] K. Murakami and H. Taguchi. Gesture recognition using recurrent neural networks. In *CHI '91 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 237–242, 1991.

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

[NM98] H. Noman and T. Miyasato. Design for locomotion interface in a large scale virtual environment - atlas : Atr locomotion interface for active self motion. In *Proceedings of ASME-DSC*, pages 111–118, 1998.

[NSM00] Hauro Noma, Toshiaki Sugihara, and Tsutomu Miyasato. Development of ground surface simulator for tel-e-merge system. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 217–224, 2000.

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

[OJN01] Dan R. Olsen Jr. and Travis Nielsen. Laser pointer interaction. In *Proceedings of CHI'01*, pages 17–22, 2001.

[Ope] OpenMASK. http ://www.irisa.fr/siames/openmask/. Web site.

[PB95] Randy Pausch and Tommy Burnette. Navigation and locomotion in virtual worlds via flight into hand-held miniature. In *SIGGRAPH 95 Conference Proceedings*, pages 399–400, 1995.

[PBWI96] Ivan Poupyrev, Mark Billinghurst, Suzanne Weghorst, and Tadao Ichikawa. The go-go interaction technique : Non-linear mapping for direct manipulation in vr. In *Proceedings of ACM Symposium on User Interface Software and Technology*, pages 79–80, 1996.

[PCvDR99] Jeffrey S. Pierce, Matthew Conway, Maarten van Dantzich, and George Robertson. Toolspaces and glances : Storing, accessing, and retrieving objects in 3d desktop applications. In *Proceedings of the 1999 Symposium on Interactive 3D graphics*, pages 163–168, 1999.

[Pec01] Choon Hong Peck. Useful parameters for the design of laser pointer interaction techniques. In *Proceedings of CHI'01*, pages 461–462, 2001.

[Per98] Ken Perlin. Quikwriting - continuous stylus-based text entry. In *ACM Symposium on User Interface Software and Technology*, pages 215–216, 1998.

[PFC+97] Jeffrey S. Pierce, Andrew Forsberg, Matthew J. Conway, Seung Hong, Robert Zeleznik, and Mark R. Mine. Image plane interaction techniques in 3d immersive environments. In *Proceedings of 1997 Symposium on Interactive 3D Graphics*, pages 39–43, 1997.

[PMAI03] Gian Pangaro, Dan Maynes-Aminzade, and Hiroshi Ishii. The actuated workbench : Computer-controlled actuation in tabletop tangible interfaces. In *ACM Symposium on User Interface Software and Technology*, 2003.

[PR97] J. Peters and U. Reif. The simplest subdivision scheme for smoothing polyhedra. *ACM Trans. Graph.*, 16(4), 1997.

[PSP99] Jeffrey S. Pierce, Brian C. Stearns, and Randy Pausch. Voodoo dolls : Seamless interaction at multiple scales in virtual environments. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*, pages 141–146, 1999.

[PTW98] Ivan Poupyrev, Numada Tomokazu, and Suzanne Weghorst. Virtual notepad : Handwriting in immersive vr. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, 1998.

[Qas97] Safwan Qasem. *Conception et Réalisation d'une Interface 3D Pour Cabri-Géomètre*. PhD thesis, Université Joseph Fourier - Grenoble, 1997.

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

[Rhe91] Howard Rheingold. *Virtual Reality*. Simon & Schuster, Inc., 1991.

[RKW01] Sharif Razzaque, Zachariah Kohn, and Mary C. Whitton. Redirected walking. In *Proceedings of Eurographics Workshop*, 2001.

[RL01] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *Proceedings of the Third Intl. Conf. on 3D Digital Imaging and Modeling*, pages 145–152, 2001.

[SB05] Ludovic Sternberger and Dominique Bechmann. Deformable ray-casting interaction technique. In *YoungVR February*, 2005.

[SBB03] Ludovic Sternberger, Dominique Bechmann, and Sylvain Brandel. Étude des métaphores d'interaction 3*d* sur le workbench, 2003.

[Sch03] Jérôme Schmidt. Interaction 3d avec des gants de données. Technical report, Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection, 2003.

[SCP95] Richard Stoakley, Matthew J. Conway, and Randy Pausch. Virtual reality on a WIM : Interactive worlds in miniature. In *Proceedings of CHI'95*, 1995.

[SEa98] Dieter Schmalstieg and L. Miguel Encarnação. A transparent personal interaction panel for the virtual table. Technical Report 10(5), Fraunhofer Center for Research in Computer Graphics (CRCG) in Providence, USA, 1998.

[SEaS99] Dieter Schmalstieg, L. Miguel Encarnação, and Zsolt Szalavári. Using transparent props for interaction with the virtual table. In *Symposium on Interactive 3D Graphics*, pages 147–153, 1999.

[SG94] Chris Shaw and Mark Green. Two-handed polygonal surface design. In *ACM Symposium on User Interface Software and Technology*, pages 205–212, 1994.

[SG97] Zsolt Szalavári and Michael Gervautz. Using the personal interaction panel for 3d interaction. In *Proceedings of the Conference on Latest Results in Information Technology*, pages 3–6, 1997.

[SHR+92] Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, D. Brookshire Conner, and Andries van Dam. Using deformations to explore 3d widget design. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH'92 Proceedings)*, volume 26, pages 351–352, 1992.

[SIW+02] E. Sharlin, Y. Itoh, Y. Watson, S. Sutphen, and L. Liu. Cognitive cubes : a tangible user interface for cognitive assessment. In *Proceedings of Human Factors in Computing Systems CHI*, pages 347–354, 2002.

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

[SPH+95] Luis Serra, Timothy Poston, Ng Hern, Chua Beng Choon, and John A. Waterworth. Interaction techniques for a virtual workspace. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology*, pages 79–80, 1995.

[SSK01] Yoichi Sato, Makiko Saito, and Hideki Koike. Real-time input of 3d pose and gestures of a user's hand and its applications for hci. *IEEE Virtual Reality*, pages 79–86, 2001.

[SSS99] G. Smith, T. Salzman, and W. Stuerzlinger. 3d scene manipulation with constraints. *Virtual and Augmented Architecture*, pages 35–46, 1999.

[SSS02] Stanislas L. Stoev, Dieter Schmalstieg, and Wolfgang Straßer. The through-the-lens metaphor - taxonomy and application. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, 2002.

[SU94] Mel Slater and Martin Usoh. Body centred interaction in immersive virtual environments. *Artificial Life and Virtual Reality*, pages 125–148, 1994.

[SUS95] Mel Slater, Martin Usoh, and Anthony Steed. Taking steps : the influence of a walking technique on presence in virtual reality. *ACM Transactions on Computer-Human Interaction*, 2(3) :201–219, 1995.

[SVDSKVDM01] Martijn J. Schuemie, Peter Van Der Straaten, Merel Krijn, and Charles A.P.G. Van Der Mast. Research on presence in vr : a survey. *CyberPsychology & Behavior*, 4(2), 2001.

[SW00] Ralf Salomon and John Weissmann. Evolutionary tuning of neural networks for gesture recognition. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 1528–1534, 2000.

[SZP89] D. J. Sturman, D. Zeltzer, and S. Pieper. Hands-on interaction with virtual environments. In *ACM Symposium on User Interface Software and Technology*, pages 19–24, 1989.

[TAC00] T. Trilling, R. Allen, and S. Channac. The role of requirements, specifications, and implementation in constructing dynamic figures. *Journal of Computers in Mathematics and Science Teaching*, 19(3) :195–209, 2000.

[TBBB02] Damien Touraine, Patrick Bourdot, Yacine Bellik, and Laurence Bolot. A framework to manage multimodal fusion of events for advanced interactions within virtual environments.

In *Proceedings of the workshop on Virtual environments 2002*, pages 159–168, 2002.

[TCH+03] N. Tarrin, S. Coquillart, S. Hasegawa, Bouguila L., and M. Sato. The stringed haptic workbench : a new haptic workbench solution. In *Proceedings of Eurographics Workshop*, 2003.

[TDS99] James M. Templeman, Patricia S. Denbrook, and Linda E. Sibert. Virtual locomotion : Walking in place through virtual environments. In *Presence*, pages 598–617, 1999.

[TJ00] Vildan Tanriverdi and Robert J. K. Jacob. Interacting with eye movements in virtual environments. In *Proceedings of CHI'00*, pages 265–272, 2000.

[TK99] Daniel Thalmann and Marcelo Kallmann. Direct 3d interaction with smart objects. In *ACM Virtual Reality Software and Technology*, pages 124–130, 1999.

[Too] MR ToolKit. www.cs.ualberta.ca/g̃raphics/mrtoolkit/. Web site.

[TS98] E. Tsang and H. Sun. An efficient posture recognition method using fuzzy logic. *Journal of Virtual Reality*, pages 112–119, 1998.

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

[VL98] N. Van Labeke. Calques 3d : a microworld for spatial geometry learning. In *ITS'98 - System Demonstrations*, 1998.

[VL99] Nicolas Van Labeke. *Prise en compte de l'usager enseignant dans la conception des EAIO*. PhD thesis, Université Henri Poincaré Nancy, 1999.

[VRC] VRCo. http ://www.vrco.com/cavelib/. Web site.

[VRJ] VRJuggler. http ://vrjuggler.org/. Web site.

[VS95] Erick Von Schweber. Virtual reality - virtually here, 1995.

[vtk] vtkVR. http ://staff.science.uva.nl/m̃scarpa/vtkvr/. Web site.

[WBV+99] Greg Welch, Gary Bishop, Leandra Vicci, Stephen Brumback, Kurtis Keller, and D'nardo Colucci. The hiball tracker : High-performance wide-area tracking for virtual and augmented environments. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology*, pages 1–11, 1999.

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

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

[Whi03] Mary C. Whitton. Making virtual environments compelling. *Communications of the ACM*, 46(7) :40–47, 2003.

[Wis01] Michael Wissen. Implementation of a laser-based interaction technique for projection screens. ERCIM NEWS, Number 46, P.31-32, 2001.

[WO90] Colin Ware and Steven Osborne. Exploration and virtual camera control in virtual three dimensional environments. In *Proceedings of ACM Symposium on Interactive 3D Graphics*, pages 175–183, 1990.

[WPA96] Maxwell J. Wells, Barry N. Peterson, and Jason Aten. The virtual motion controller : A sufficient-motion walking simulator. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 1–8, 1996.

[WR00] Colin Ware and Jeff Rose. Rotating virtual objects with real handles. *ACM Transactions on Computer-Human Interaction*, 6(2) :162–180, 2000.

[WS03] Daniel Wagner and Dieter Schmalstieg. First steps towards handheld augmented reality. In *Proceedings of the 7th International Conference on Wearable Computers*, 2003.

[ZBM94] Shumin Zhai, William Buxton, and Paul Milgram. The "silk cursor" : Investigating transparency for 3d target acquisition. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 1, pages 459–464, 1994.

[ZHH96] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch : An interface for sketching 3d scenes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 163–170. Addison Wesley, 1996.

[ZLJAFK02] Robert C. Zeleznik, Joseph J. LaViola Jr., Daniel Acevedo Feliz, and Daniel F. Keefe. Pop through button devices for ve navigation and interaction. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, 2002.

[Zor05] D. Zorin. Modeling with multiresolution subdivision surfaces. *Tutorial Eurographics*, 2005.

Table des figures

Liste des tableaux