CHRISTIAN DOMPIERRE

Avatar :

Une application de réalité virtuelle utilisable comme nouvel outil de mise en scène collaborative

Mémoire présenté
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de maîtrise en génie électrique
pour l'obtention du grade de Maître ès sciences

FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

2007

Résumé

Une des plus importantes lacunes des outils actuels de mise en scène est que la collaboration en temps réel est très difficile à réaliser. En fait, les metteurs en scène doivent actuellement être situés au même endroit pour collaborer, ce qui n'est pas toujours possible.

Les logiciels actuels de mise en scène ne permettent généralement pas la collaboration en temps réel. De plus, ces systèmes sont souvent très dispendieux et complexes d'utilisation.

Une solution plus adaptée combinant des concepts de réalité virtuelle, technologie réseaux et vision numérique a ainsi été spécifiquement développée pour combler ces lacunes.

Ce mémoire présente l'application de réalité virtuelle permettant la mise en scène collaborative résultant de nos travaux de recherche.

Les défis les plus importants étaient d'assurer une synchronisation en temps réel entre une scène réelle et une scène virtuelle partagée et de minimiser les coûts tout en offrant un système multi-plates-formes et convivial.

Abstract

One of the more important limitations of actual tools for performing arts design is that collaboration between designers is hard to achieve. In fact, designers must actually be co-located to collaborate in the design of a show, something that is not always possible.

Actual software tools for performing arts design do not generally provide real-time collaboration and are not really convenient for collaborative work. In addition, these systems are often expensive and complex to operate.

A more adapted solution combining concepts from virtual reality, network technology, and computer vision has then been specifically developed to solve these issues.

This master thesis presents the virtual reality application for supporting distributed collaborative production of theater shows resulting from our research.

Challenges were to ensure real-time synchronization between a real and a shared virtual scene and to keep system cost as low as possible while offering platform independence and user-friendliness.

Avant-propos

Au départ, il y eut un rêve. Ce rêve, je le décrivais alors comme le cinéma du futur, le « cinéma virtuel ». Loin d'en saisir toutes les implications, je rêvais de rendre l'expérience cinématographique plus immersive par l'emploi de la réalité virtuelle.

Vinrent ensuite les encouragements de mes proches qui m'incitèrent et me donnèrent la motivation nécessaire à me lancer dans le projet. Afin de me familiariser avec cette technologie et d'aller chercher les compétences nécessaires à l'aboutissement de ce projet, je décidai alors de m'inscrire à la maîtrise.

Sans me permettre d'atteindre mon objectif de départ, cette maîtrise me permit tout de même de me familiariser avec cet univers extraordinaire qu'est la réalité virtuelle. Ce mémoire ainsi que l'application qu'il présente sont les fruits de mon travail sur ce sujet. Bien qu'il fut fort intéressant, l'achèvement de ce travail n'aurait jamais été possible sans le support continu de mes parents et de toute ma famille. Pour cela, je les remercie.

Je tiens aussi à remercier Denis Laurendeau, mon directeur de recherche ainsi que Alexandra Branzan Albu, ma co-directrice. Cette maîtrise n'aurait évidemment pas été possible sans leur expérience et leur soutien. Je remercie aussi les chercheurs du LVSN, du LANTISS et du Laboratoire de robotique de l'Université Laval qui ont mis sur pied le projet de Castelet Électronique ainsi que le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG) et le Conseil des Arts du Canada (CAC) qui ont supporté ce projet par une subvention stratégique.

Je remercie de plus le LVSN pour toutes les ressources mises à ma disposition. Merci à Annette Schwerdtfeger pour son aide lors de la rédaction en anglais et merci à Denis Ouellet et Sylvain Comtois pour leur aide sur le plan informatique. Merci aussi à Robert Gardiner de m'avoir permis d'utiliser ses images dans mon mémoire.

Finalement, je remercie Simon, Jean-Daniel, Philippe et Nicolas pour leur aide, leurs conseils et leur support tout au long de cette aventure.

Je dédie ce mémoire à Rollande et François,
mes très chers parents.

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Entrée en matière

En mars 2004, l'Université Laval inaugura le *Laboratoire des nouvelles technologies de l'image, du son et de la scène* (LANTISS). Ce laboratoire est en fait le fruit de l'association d'entités provenant de divers domaines, à savoir le *Laboratoire de vision et systèmes numériques* (LVSN) [1], le *Laboratoire de robotique* [2], le *Centre d'optique photonique et laser* (COPL) [3], le *Centre d'artistes Avatar* [4] (membre de la coopérative *Méduse*) ainsi que le groupe multidisciplinaire *Ex Machina* [5] du metteur en scène Robert Lepage. La mission première du LANTISS est d'encourager le mariage entre Arts et nouvelles technologies. Dans cette optique, le LANTISS a entre autres mis sur pied un projet nommé *Castelet Électronique*[1]. Ce projet, supervisé par Clément Gosselin du Laboratoire de robotique et supporté par une subvention stratégique CRSNG-CAC, implique des gens de tous les groupes de recherche formant le LANTISS. Le but du Castelet Électronique est le développement d'un outil sans précédent permettant l'utilisation de procédés technologiques avant-gardistes dans le processus de mise en scène traditionnel.

[1] Le terme *castelet* réfère à ce qui est plus communément appelé un théâtre de marionnettes.

1.2 Mise en contexte

1.2.1 La mise en scène

Avant qu'une œuvre théâtrale ou cinématographique ne soit présentée au grand public, celle-ci doit faire l'objet d'une direction artistique. Lors de ce processus, chacune des scènes de l'œuvre est décortiquée, analysée et travaillée méticuleusement. C'est ce que l'on appelle la *mise en scène* de l'œuvre en question.

1.2.2 Les sketchs

Les metteurs en scène disposent de divers outils pour les aider à faire leur travail. Traditionnellement, ceux-ci utilisent généralement deux outils pour mettre au point une scène. Premièrement, les scènes sont normalement dessinées à la main pour donner ce que l'on appelle les *sketchs* de ces scènes. La figure 1.1 présente un exemple de sketch tiré de la mise en scène de la pièce *Lillies* présentée au *Arts Club Theatre* à Vancouver en 1994.

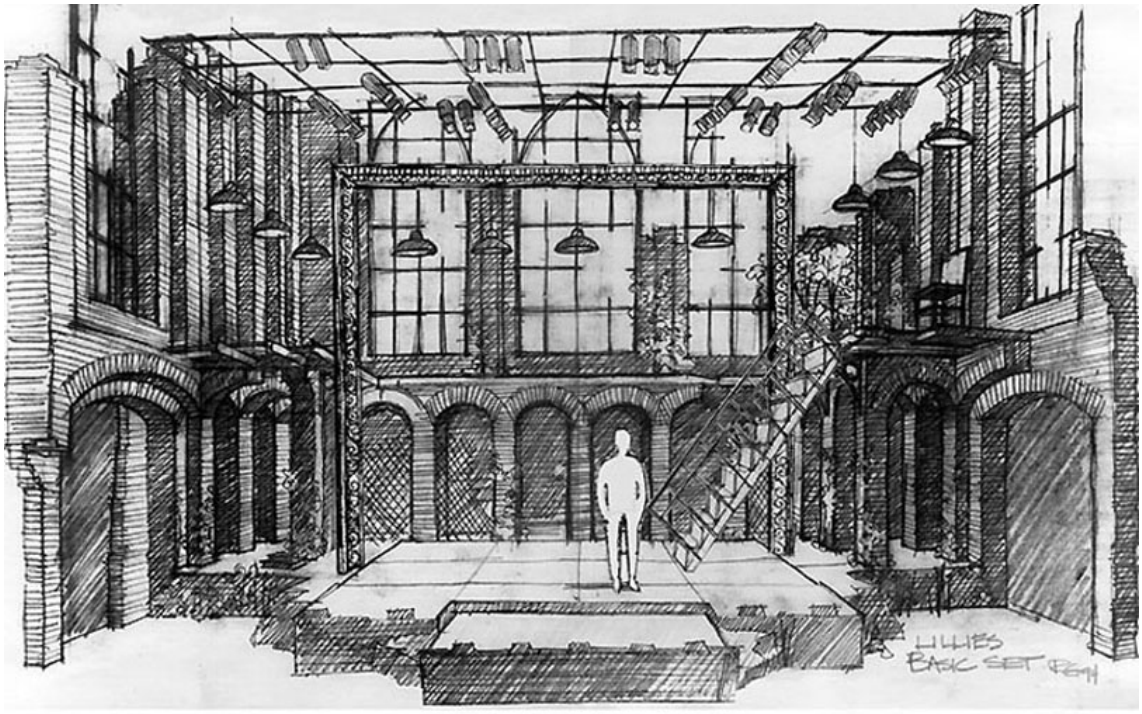


FIG. 1.1 – Sketch effectué par Robert Gardiner pour la mise en scène de la pièce *Lillies* présentée au *Arts Club Theatre* à Vancouver en 1994.

L'utilisation des sketchs est pratiquement essentielle au travail des metteurs en scène. Principalement, cet outil leur permet de visualiser une idée de scène sans avoir à la construire physiquement. Ils peuvent ainsi évaluer et même faire évoluer cette idée sans avoir à dépenser temps et argent dans des constructions physiques. Les sketchs constituent donc un outil très intéressant pour les metteurs en scène.

Cet outil simple d'utilisation et peu coûteux nécessite cependant un certain talent et présente aussi quelques lacunes. Premièrement, ce genre de dessin peut parfois être assez fastidieux et long à produire. De plus, l'interaction avec les éléments constitutifs de la scène est impossible. Ainsi, chaque changement apporté à la scène, aussi mineur soit-il, nécessite l'élaboration d'un autre sketch. Cela entraîne souvent la nécessité de produire une très grande quantité de sketchs. À titre d'exemple, la figure 1.2 présente un employé de *Pixar* exposant son idée de film d'animation. On peut y voir une partie de tous les sketchs qui seront éventuellement nécessaires à la mise en scène.

FIG. 1.2 – Exemple du grand nombre de sketchs souvent nécessaires à une mise en scène (Pixar).

Chacun de ces sketchs étant à la base assez long à produire, la conception de tous les sketchs nécessaires à la mise en scène d'une pièce ou d'un film peut être extrêmement longue. Aussi, bien qu'il soit possible d'illustrer une scène tridimensionnelle à l'aide d'un sketch, le caractère 2D de ce dernier n'est pas idéal pour évaluer pleinement une idée de scène. En effet, la perspective peut parfois être trompeuse sur la profondeur nécessaire à une scène.

Un sketch peut donner une bonne idée globale de ce dont aura l'air une scène une fois construite et peut être très utile durant le processus de conception, mais les lacunes tout juste exposées amènent les metteurs en scène à utiliser un autre outil conjointement aux sketchs : la maquette.

1.2.3 Les maquettes

Le terme *maquette* désigne en fait les modèles réduits de scène que les metteurs en scène utilisent pour faire leur travail. La figure 1.3 présente un exemple de maquette tirée de la mise en scène de la pièce *Anne of Green Gables* présentée au *Seattle Children's Theatre* en 1992.

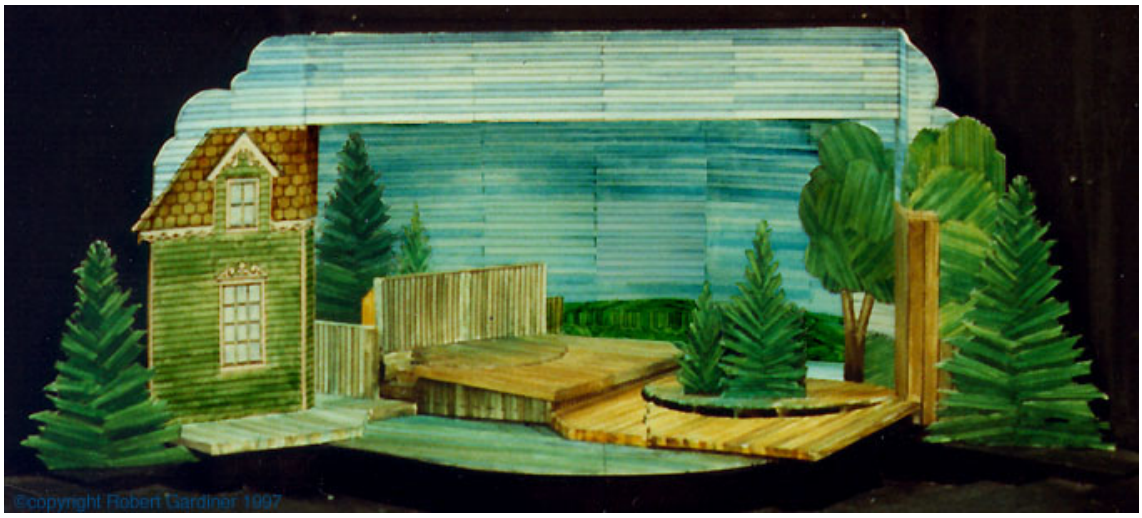


FIG. 1.3 – Maquette confectionnée par Robert Gardiner pour la mise en scène de la pièce *Anne of Green Gables* présentée au *Seattle Children's Theatre* en 1992.

Les maquettes peuvent être vues comme un outil complémentaire aux sketchs. En effet, les maquettes comblent une certaine partie des lacunes présentes chez les sketchs. Premièrement, l'interaction avec les éléments constitutifs de la scène est maintenant possible par l'utilisation des maquettes. Ainsi, les changements mineurs apportés à la scène comme le déplacement ou la suppression d'un élément constitutif qui nécessitaient la conception de nouveaux sketchs peuvent maintenant tous se faire directement sur la maquette. Il n'est donc pas nécessaire de faire une nouvelle maquette chaque fois qu'un nouveau sketch serait requis. Cela augmente la productivité et diminue grandement la charge de travail. Deuxièmement, contrairement aux sketchs, les maquettes bénéficient des 3 dimensions de l'espace. La profondeur des scènes peut donc être beaucoup plus justement évaluée par une maquette que par un sketch.

Cependant, les maquettes ne constituent pas pour autant un outil parfait et présentent d'ailleurs aussi plusieurs lacunes. D'abord, tout comme pour les sketchs, la conception d'une maquette nécessite un certain talent et peut parfois être une tâche fastidieuse. Les éléments constitutifs d'une scène ne sont pas toujours disponibles en modèle réduit. Ainsi, il est souvent nécessaire de concevoir ces éléments sur mesure. Cette procédure peut parfois demander beaucoup de temps. De plus, contrairement

aux sketchs, l'archivage des maquettes nécessite beaucoup d'espace[2].

Le fait même d'utiliser une échelle réduite entraîne aussi certains problèmes. Cela diminue par exemple la qualité de l'immersion et donc la facilité de visualisation. En effet, bien que les maquettes comme celle présentée à la figure 1.3 peuvent grandement aider les metteurs en scène à visualiser leurs idées, celles-ci exigent par contre que ces derniers usent de leur imagination. C'est que les maquettes présentent généralement les défauts d'être incomplètes et de ne pas être très réalistes, que ce soit au niveau des acteurs, du décor ou de l'éclairage. Par exemple, la géométrie tridimensionnelle de la scène n'est souvent qu'approximée grossièrement par la maquette. Les metteurs en scène ne sont donc jamais certains du résultat tant que la scène grandeur réelle n'est pas construite.

Ces défauts font en sorte que la visualisation à partir d'une maquette, quoique plus facile qu'à l'aide d'un sketch, n'est pas toujours triviale et peut même parfois être trompeuse.

Une autre lacune de la maquette est sa piètre qualité d'immersion. En effet, encore à cause de l'échelle réduite, il est plutôt difficile de se sentir immergé dans l'univers représenté par la maquette. De plus, lorsque l'on pense à la mise en scène d'œuvres cinématographiques, il n'est parfois pas possible avec une maquette de prévoir l'allure finale d'une scène. Par exemple, il est parfois impossible, à l'aide d'une maquette, d'avoir même un simple aperçu de ce que la caméra (et donc le téléspectateur) verra. Le résultat final n'est dans ce cas connu qu'après le tournage et peut alors apporter des surprises aux directeurs artistiques.

Finalement, une des principales lacunes des maquettes mise en évidence par les metteurs en scène du LANTISS est l'impossibilité de collaborer à une mise en scène à distance. Ainsi, les metteurs en scène voulant collectivement participer à une mise en scène doivent nécessairement travailler ensemble, au même endroit, au même moment.

Les sketchs et les maquettes présentent donc plusieurs lacunes et défauts, mais les metteurs en scène ont tout de même avantage à faire abstraction de ceux-ci. En effet, malgré leurs points faibles, ces outils permettent aux metteurs en scène de visualiser les scènes qu'ils ont en tête sans avoir à les construire en grandeur réelle, ce qui est souvent très coûteux et donc non souhaitable lorsque ces scènes ne sont pas encore au point et qu'elles sont encore très fortement susceptibles d'être modifiées. Ces outils sont donc très utiles et très appréciés par les metteurs en scène. Ceci étant dit, des améliorations

[2] À moins que l'on dispose d'un modèle CAD de la maquette. Ce sujet est d'ailleurs traité à la section 1.2.4.

étaient envisageables et celles-ci allaient venir du monde informatique.

1.2.4 Les logiciels de modélisation et de rendu 3D

Depuis déjà bon nombre d'années, et plus particulièrement depuis l'arrivée des cartes graphiques[3], les ordinateurs permettent de générer des images d'une qualité graphique très impressionnante. Certains logiciels ([6], [8]) utilisant des techniques de rendu telles le *raytracing* ou le *global illumination* arrivent à générer des images d'un réalisme à s'y méprendre (voir figure 1.4). Chaque détail peut être pris en considération : le nombre de sources de lumière, leur type (ponctuel, directionnel, ...), leur position dans l'espace, la géométrie des objets, des murs, des acteurs, leurs couleurs, textures, propriétés de réflectance, leur position dans l'espace, etc. Cela permet de générer des scènes réalistes, complètes et précises, ce qui est justement pratiquement impossible à réaliser avec une maquette.

FIG. 1.4 – Image artificielle à caractère réaliste générée à l'aide du logiciel *POV-Ray* [6].

Il existe un bon nombre de logiciels de modélisation et de rendu 3D sur le marché

[3]Le processus de rendu graphique, incluant entre autres le travail effectué par les cartes graphiques, est exposé en détail à la section 2.2.1 en page 27.

([6], [7], [8]). Du point de vue de la mise en scène, ce genre de logiciel peut en fait combler quelques lacunes des outils plus traditionnels de mise en scène. Premièrement, ces logiciels permettent de créer des images beaucoup plus complètes et réalistes que les sketchs. La figure 1.5 présente par exemple une image tirée de la mise en scène de la pièce *Unity, 1918* présentée au *Touchstone Theatre* à Vancouver en 2001. Cette image fut générée par le metteur en scène Robert Gardiner à l'aide du logiciel *VectorWorks* [9].

FIG. 1.5 – Image générée par Robert Gardiner à l'aide du logiciel *VectorWorks* pour la mise en scène de la pièce *Unity, 1918* présentée au *Touchstone Theatre* à Vancouver en 2001.

En plus d'être plus complètes et plus réalistes que les sketchs, les images créées à l'aide des logiciels de modélisation et de rendu 3D ne présentent plus de problème d'échelle. Ces logiciels offrent en fait un espace théoriquement infini. Ainsi, les scènes créées à l'aide de ces logiciels peuvent être parfaitement à l'échelle, peu importe leur taille, ce qui est parfois très difficile, voire impossible au niveau des maquettes et des sketchs. Ainsi, malgré le fait qu'il soit présenté en 2D sur l'écran de l'ordinateur, cet outil apporte une visualisation souvent plus révélatrice de l'allure géométrique des scènes.

De plus, ce type de logiciel, contrairement aux maquettes, permet aux metteurs en scène cinématographiques de se positionner littéralement n'importe où et de pouvoir ainsi observer l'image qui serait captée par une caméra située à tel ou tel endroit.

Un autre avantage des logiciels de modélisation et de rendu 3D est qu'ils sont des outils de travail très flexibles permettant d'effectuer des changements très rapidement et à faible coût. Il est par exemple possible de changer instantanément la couleur des murs d'une pièce ou encore le style des meubles ou accessoires sans avoir à redessiner un nouveau sketch ou à refaire une nouvelle maquette. Il existe à cet effet une panoplie de modèles 3D dont les metteurs en scène peuvent bénéficier. Certains sites Internet en offrent en téléchargement payant [10] [11] ou même gratuit [12] [13]. Les metteurs en scène peuvent donc se bâtir une banque de modèles 3D de plus en plus complète, laquelle, grâce à son caractère informatique, peut être entièrement stockée sur un disque dur et ne prendre pratiquement aucun espace, contrairement aux maquettes dont l'archivage requiert souvent un entrepôt.

Tout comme les maquettes, mais contrairement aux sketchs, les logiciels de modélisation et de rendu 3D offrent la possibilité d'interagir en temps réel avec les éléments constitutifs de la scène. Cependant, cette interaction est moins intuitive et moins triviale que dans le cas des maquettes. En effet, avec ce type de logiciel, le metteur en scène est limité aux 2 dimensions de l'écran de son ordinateur pour interagir avec les éléments de sa scène 3D.

L'utilisation d'un logiciel de modélisation et de rendu 3D n'est cependant pas contradictoire à l'utilisation d'une maquette et ces deux outils peuvent en fait être complémentaires tels que le sont souvent sketchs et maquettes. Ainsi, afin de profiter des 3 dimensions de l'espace pour d'une part interagir plus intuitivement avec les éléments constitutifs de la scène et d'autre part pour mieux évaluer la géométrie de la scène, les metteurs en scène construisent souvent des maquettes à partir de leurs modèles issus de logiciels de modélisation et de rendu 3D. Ces logiciels permettent la construction de maquettes sans erreur d'échelle et beaucoup plus réalistes et complètes que celles qui peuvent être élaborées à partir de sketchs à la main. La figure 1.6 présente une photo de la maquette obtenue du modèle 3D qui fut élaboré pour la mise en scène de la pièce *Unity, 1918* et dont une image fut présentée à la figure 1.5.

Même si les maquettes construites sur la base de ces modèles 3D présentent encore certaines des lacunes énoncées à la section précédente, l'utilisation des logiciels de modélisation et de rendu 3D permet de régler une bonne partie des lacunes des outils de mise en scène. Cependant, les logiciels de ce type sont souvent d'une grande complexité. Leur utilisation requiert la plupart du temps une certaine expertise. Cette non convivialité fait en sorte que même si cet outil peut être très utile aux metteurs en scène, son utilisation au niveau des professionnels des arts de la scène n'est pas très répandue. Le coût souvent élevé de ces applications vient d'ailleurs amplifier cette situation.

FIG. 1.6 – Maquette créée par Robert Gardiner à partir d'un modèle 3D généré à l'aide du logiciel *VectorWorks* pour la mise en scène de la pièce *Unity, 1918* présentée au *Touchstone Theatre* à Vancouver en 2001.

De plus, certaines lacunes des outils plus traditionnels de mise en scène comme la mauvaise qualité d'immersion due à l'échelle réduite ne peuvent pas être réglées par l'utilisation de ce type de logiciel. Aussi, cette technologie ne permet de régler que partiellement la lacune que présentent les maquettes au niveau de la collaboration. En effet, bien qu'il soit maintenant possible de collaborer à une mise en scène en échangeant des fichiers informatiques, cette collaboration ne peut se faire en temps réel et les metteurs en scène doivent donc travailler à tour de rôle.

Ainsi, malgré le fait que cette technologie règle plusieurs des lacunes précédemment identifiées des outils plus traditionnels de mise en scène, certaines limitations restent toujours présentes. Cependant, les avancées technologiques des dernières années permettent maintenant d'envisager de surmonter ces difficultés par l'utilisation d'un nouveau médium appelé *réalité virtuelle*.

1.2.5 La réalité virtuelle

Un des concepts fondamentaux dans le domaine de la réalité virtuelle est celui de *monde virtuel*. Un monde virtuel est défini comme un monde imaginaire n'existant pas physiquement[4], mais étant tout de même régi par un ensemble de lois lui étant propres. Le monde imaginaire d'un rêve est un bon exemple de monde virtuel.

Les mondes virtuels sont des mondes imaginaires qui sont indépendants du médium de transmission utilisé pour les représenter. D'ailleurs, un même monde virtuel peut être représenté de diverses façons, que ce soit par un film, un livre ou une peinture par exemple. Les logiciels de rendu 3D discutés à la section précédente sont d'ailleurs une autre façon de visualiser des mondes virtuels.

Bien qu'il lui soit possible d'interagir avec ceux-ci, l'utilisateur des logiciels de modélisation et de rendu 3D observe cependant toujours ces mondes virtuels de l'extérieur et n'est jamais pleinement immergé dans ceux-ci.

À cet effet, les dernières années ont vu l'émergence d'un nouveau médium appelé *réalité virtuelle* (RV) qui est souvent définie de façon simpliste comme l'immersion (c'est-à-dire la sensation d'être présent) dans un monde artificiel (virtuel).

Cette définition vient du fait qu'en RV, l'utilisateur visualise le monde virtuel en 3D et en grandeur réelle, exactement comme il le verrait si ce monde existait réellement et qu'il y était présent.

Cette définition n'est cependant pas complète car en plus de permettre l'immersion, la RV fournit une rétroaction sensorielle. En d'autres termes, les stimuli sensoriels (la plupart du temps visuels) présentés à l'utilisateur d'un système de RV sont fonction de la position et de l'orientation (et donc des déplacements) de ce dernier dans l'espace. Ainsi, l'utilisateur a l'impression d'être réellement présent dans le monde virtuel plutôt que de simplement assister à une présentation immersive. La définition précédente est aussi incomplète en ce qu'elle n'inclut pas le fait que le monde virtuel est interactif et que l'interaction avec les éléments constitutifs de ce monde virtuel est beaucoup plus intuitive et naturelle que dans un logiciel de modélisation et de rendu 3D.

[4]Un monde virtuel n'est pas nécessairement associé à un support physique à proprement parler. En effet, l'existence d'un monde virtuel ne nécessite aucunement l'existence d'un monde physique lui étant associé. Cependant, un monde virtuel peut aussi bien représenter le monde réel (ou plutôt une partie du monde réel) qu'un monde totalement imaginaire. Leur caractère intangible ne limite en rien leur aptitude à représenter les choses du monde réel.

Sherman et Craig [33] proposent une définition de la réalité virtuelle beaucoup plus complète et juste : ***Virtual reality*** *: a medium composed of interactive computer simulations that sense the* participant's *position and actions and replace or augment the feedback to one or more senses, giving the feeling of being mentally immersed or present in the simulation (a virtual world).*

La RV permet donc de créer un monde virtuel et de s'immerger à l'intérieur de celui-ci afin d'y vivre une expérience très similaire à ce que l'on vivrait si ce monde existait réellement et que l'on y était présent. Cependant, le but de la RV n'est pas de créer un environnement virtuel simulant parfaitement le réel, mais plutôt de créer des environnements virtuels *semblant* réels. La RV offre ainsi des possibilités très intéressantes que le monde réel n'offre pas. Contrairement à l'environnement réel, l'environnement virtuel est totalement contrôlable et entièrement modifiable. Il est en fait possible de contrôler entièrement le fonctionnement et l'apparence de l'environnement virtuel, ce qui est sans contredit très intéressant d'un point de vue créatif. La RV est donc tout indiquée pour tout ce qui se rapproche du prototypage (comme la mise en scène) puisqu'elle permet de valider ou d'infirmer une idée et même de modifier un prototype avant de le construire réellement.

Les récents développements dans le domaine des périphériques spéciaux nécessaires à la RV permettent maintenant d'envisager l'utilisation de ce médium à divers propos. En particulier, il est maintenant raisonnable de songer à intégrer cette technologie dans le domaine de la mise en scène pour laquelle la RV est le médium idéal.

De fait, cette technologie présente tous les avantages des outils traditionnels de mise en scène, mais sans leurs défauts. La RV comble même toutes les lacunes de chacun de ces outils. Pouvant être vue comme une extension des logiciels de modélisation et de rendu 3D, la RV présente forcément tous les bénéfices de ces outils (réalisme, complétude, scène à l'échelle, espace infini, archivage, etc.). La RV comble de plus toutes les lacunes que ces outils peuvent présenter. Par exemple, le monde virtuel est maintenant visualisé en 3D comblant ainsi la lacune des logiciels de modélisation et de rendu 3D de limiter la visualisation à 2 dimensions, ce qui entraînait souvent la construction d'une maquette, laquelle est maintenant inutile. De plus, tous les problèmes reliés à l'échelle réduite des mondes virtuels présentés à l'écran de l'ordinateur ou dans la maquette sont maintenant chose du passé puisque le monde virtuel est visualisé en grandeur réelle. Cette possibilité est un avantage marqué qu'aucun des outils actuels de mise en scène n'offre. La RV règle aussi le problème de la complexité d'interaction avec les éléments du monde virtuel dans les logiciels de modélisation et de rendu 3D en permettant une interaction aussi naturelle et triviale que celle présente avec les maquettes.

Tout comme les logiciels de modélisation et de rendu 3D, la RV permet, grâce à sa simplicité et à sa possibilité de changement rapide, d'augmenter la productivité et de diminuer la charge de travail des metteurs en scène. La RV offre aussi la possibilité aux metteurs en scène cinématographiques de positionner leur caméra virtuelle littéralement n'importe où et de voir immédiatement ce que la caméra capterait.

Finalement, un des gros avantages d'utiliser la RV comme outil de mise en scène vient des possibilités qu'elle offre au niveau de la collaboration. En effet, il est possible d'immerger plusieurs utilisateurs dans un même monde virtuel sans que ces derniers ne soient physiquement dans le même lieu. Il est ainsi possible pour plusieurs metteurs en scène situés en des lieux différents de travailler sur une même mise en scène simultanément par le biais de la RV.

Les tableaux des figures 1.7 et 1.8 résument les lacunes et les atouts des différents outils abordés jusqu'ici. Tel qu'on peut le voir dans ces tableaux, la RV présente tous les avantages que l'on peut rechercher chez un outil de mise en scène, mais aucun des inconvénients des outils actuels. Cette technologie est donc tout indiquée pour servir de nouvel outil pour les metteurs en scène. C'est cette constatation qui est à l'origine du projet qui a mené au présent mémoire.

Lacune \ Outil	Sketchs	Maquettes	Logiciels de rendu 3D	Réalité virtuelle
Fastidieux	✖	✖		
Nécessite talent	✖	✖		
Archivage nécessitant beaucoup d'espace		✖		
Échelle réduite	✖	✖	✖	
Visualisation trompeuse	✖	✖	✖	
Problèmes d'échelle	✖	✖	✖	

FIG. 1.7 – Lacunes présentes chez divers outils de mise en scène.

1.3 Définition du projet

Tel que mentionné précédemment, notre projet de maîtrise s'inscrivait à l'intérieur du projet de Castelet Électronique mis sur pied par les différents groupes constituant le LANTISS. Il est donc naturel de définir ce projet de Castelet Électronique avant d'entrer dans les détails de notre projet comme tel.

Atout \ Outil	Sketchs	Maquettes	Logiciels de rendu 3D	Réalité virtuelle
Réalisme			✔	✔
Complétude			✔	✔
Précision			✔	✔
Développement rapide améliorant la productivité			✔	✔
Espace infini			✔	✔
Point de vue illimité			✔	✔
Interactivité		✔	✔	✔
Visualisation tridimensionnelle		✔		✔
Convivialité	✔	✔		✔
Grandeur réelle				✔
Immersion				✔
Collaboration				✔

FIG. 1.8 – Atouts recherchés chez les outils de mise en scène.

1.3.1 Le Castelet Électronique

Toujours dans l'esprit du LANTISS qui a pour but de favoriser le mariage entre Arts et nouvelles technologies, le principal objectif du projet de Castelet Électronique était de faire en sorte que les technologies de pointe puissent être utilisées au profit des metteurs en scène. L'idée était d'exploiter les possibilités offertes par ces technologies afin de simplifier le travail des metteurs en scène et de leur offrir de nouvelles possibilités. Cet objectif plutôt vague du projet de Castelet Électronique se précisa au fil des nombreuses rencontres impliquant des représentants de chacune des entités constituant le LANTISS et des objectifs plus clairs furent ainsi proposés.

Au terme des rencontres, ces objectifs étaient de développer deux composantes, à savoir un *castelet réel* et un *castelet virtuel*.

Le castelet réel dont il est question ici allait être essentiellement une maquette traditionnelle. Cet outil classique est un outil bien connu et apprécié par bon nombre de metteurs en scène et directeurs artistiques. Ainsi, plutôt que de priver ces derniers d'un outil qu'ils apprécient, le projet de Castelet Électronique se proposait d'augmenter ses fonctionnalités et d'élargir son champ de possibilités. En ce sens, bien que le castelet réel allait être dans son essence une maquette traditionnelle, celle-ci allait maintenant incorporer des éléments offrant des possibilités hors du commun. Plus précisément, le castelet réel allait être une maquette pourvue d'un plancher robotisé et d'un système d'éclairage à base de diodes électroluminescentes et de fibres optiques. Ces éléments allaient offrir diverses nouvelles possibilités aux metteurs en scène.

Premièrement le plancher robotisé allait permettre aux metteurs en scène de dyna-

miquement reconfigurer la scène dans leur maquette. Cette possibilité est évidemment très intéressante d'un point de vue artistique. Cependant, il aurait été inutile que cette maquette offre des possibilités qu'il est impossible de transposer à une scène réelle. Or cela n'allait pas être le cas puisque le castelet réel allait en plus avoir sa contrepartie en grandeur réelle dans les locaux du LANTISS. Celle-ci allait en fait être une scène grandeur réelle munie d'un plancher robotisé offrant les mêmes possibilités que le castelet réel.

Le système d'éclairage à base de fibres optiques dont allait aussi être muni le castelet réel allait pour sa part offrir un éclairage d'un réalisme et d'une précision incomparables à ce qui est actuellement atteint au niveau des maquettes. Ce nouveau système allait permettre aux metteurs en scène de pouvoir prototyper l'éclairage de la scène réelle beaucoup plus précisément que ce qu'il leur est présentement possible de faire avec les systèmes d'éclairage actuellement utilisés sur les maquettes.

En plus de ces innovations au niveau de la maquette, les objectifs du projet de Castelet Électronique comprenaient le développement d'une composante appelée *castelet virtuel*. Comme son nom l'indique, le castelet virtuel allait être la contrepartie virtuelle du castelet réel, donc une extension virtuelle de la maquette traditionnelle.

Le castelet virtuel n'allait cependant pas être qu'une simple réplique virtuelle du castelet réel et son ajout n'était pas sans intérêt. L'utilisation du castelet virtuel allait en fait offrir plusieurs avantages sur l'utilisation du castelet réel seul. En fait, son utilisation allait apporter tous les avantages déjà mentionnés que présente l'utilisation de la RV sur l'utilisation de maquettes. Notamment, le réalisme pratiquement impossible à atteindre avec le castelet réel allait alors être facilement envisageable avec le castelet virtuel. Les acteurs de la scène allaient par exemple pouvoir être représentés par des avatars[5] très réalistes dans le castelet virtuel. Ces derniers allaient même à la limite pouvoir être animés dans le monde virtuel. L'éclairage de la scène virtuelle allait aussi pouvoir refléter fidèlement celui de la scène finale réelle et, à l'inverse, le castelet virtuel allait même pouvoir être utilisé comme plate-forme de prototypage de cet éclairage. De plus, le décor dans le castelet virtuel allait pouvoir être aussi complet (meubles, accessoires, etc.) et fidèle (forme de la pièce, des objets, leur texture, couleur, réflectance, etc.) au décor de la scène réelle que voulu, ce qui est parfois difficile à réaliser au niveau d'une maquette comme le castelet réel. Un autre atout fort intéressant du castelet virtuel est que ce dernier allait en plus permettre de visualiser le tout en grandeur réelle. L'utilisation du castelet virtuel allait aussi avoir pour effet bénéfique de diminuer les coûts associés à la mise en scène et à la direction artistique. Les coûts associés à la

[5]Un avatar est un objet virtuel utilisé pour représenter une personne ou un objet réel (physique) dans un monde virtuel.

création des modèles réduits allaient par exemple pouvoir être entièrement éliminés avec l'utilisation du castelet virtuel. Il allait en effet maintenant être possible de créer un seul castelet réel générique pouvant être utilisé pour toute mise en scène. Le décor de ce castelet réel générique allait alors être très épuré et le décor spécifique à chaque scène allait plutôt être visualisé dans le castelet virtuel. Il allait aussi être possible de cette façon de passer instantanément d'un décor à un autre pour voir ce dont aurait l'air la scène selon une nouvelle perspective artistique ou encore de changer des éléments du décor comme la couleur des murs ou le style des accessoires d'une pièce sans coût additionnel ni manipulation fastidieuse.

Grâce à son caractère virtuel, chaque état du castelet virtuel allait pouvoir être conservé en mémoire pour fin d'archivage. Il allait ainsi être possible de consulter toutes les étapes de création ayant mené à l'aboutissement d'une scène.

Finalement, l'avantage le plus significatif de l'utilisation du castelet virtuel allait sans contredit être associé aux possibilités que ce dernier allait offrir au niveau de la collaboration. Cet outil avant-gardiste qu'est le Castelet Électronique allait en effet permettre à plusieurs directeurs artistiques physiquement distants les uns des autres de modifier simultanément le contenu du castelet virtuel et de participer ainsi collectivement à une même mise en scène. Cette possibilité sans précédent dans le domaine de la mise en scène allait évidemment ouvrir une porte sur des avenues fort intéressantes pour les artisans de ce métier.

Le castelet virtuel n'avait cependant pas pour objectif de remplacer totalement l'outil classique qu'est la maquette, mais plutôt d'être utilisé de pair avec ce dernier (le castelet réel) afin d'augmenter ses fonctionnalités et d'offrir plus de possibilités aux metteurs en scène. Le castelet réel allait ainsi servir de périphérique d'entrée à l'application de RV présentant le castelet virtuel. Les metteurs en scène allaient en fait se servir du castelet réel comme d'une maquette pour faire leur mise en scène comme ils ont l'habitude de le faire, mais allaient plutôt visualiser un résultat beaucoup plus réaliste et révélateur dans le castelet virtuel et ce, en 3D, en grandeur réelle et en temps réel.

Dans sa définition finale, le projet de Castelet Électronique était sans contredit un projet d'envergure. L'élaboration de ses différentes composantes fut donc répartie entre les diverses entités impliquées dans le projet. Ainsi, le plancher robotisé du castelet réel fut développé par un étudiant du Laboratoire de Robotique de l'Université Laval [2] alors qu'un étudiant du Centre d'Optique Photonique et Laser (COPL [3]) s'occupa du système d'éclairage. Nos travaux portèrent quant à eux sur le castelet virtuel.

1.3.2 Le projet Avatar

Le projet **Avatar** faisant l'objet de ce mémoire consistait à permettre l'utilisation de la RV dans le processus de mise en scène. Ce projet consistait en fait à permettre aux metteurs en scène du LANTISS de visualiser et d'interagir avec le castelet virtuel. Plus précisément, nous devions permettre aux metteurs en scène de visualiser et d'interagir avec les scènes virtuelles correspondant aux scènes réelles mises au point à partir du castelet réel. À l'inverse, nous devions aussi permettre aux metteurs en scène de pouvoir créer des scènes directement (et uniquement) dans le monde virtuel. Cependant, nous devions faire en sorte qu'il soit alors toujours possible de recréer exactement, sur le castelet réel, une scène ainsi virtuellement créée.

Lorsque utilisées de pair avec le castelet réel, les scènes virtuelles allaient évidemment devoir être en constante adéquation avec le contenu du castelet réel. Ainsi, les modifications apportées à la scène sur le castelet réel allaient devoir se refléter immédiatement dans le castelet virtuel. À l'inverse, les modifications apportées directement dans le castelet virtuel allaient devoir se refléter dans le castelet réel. Finalement, nous devions permettre la collaboration en temps réel entre différents metteurs en scène géographiquement distants. Plusieurs metteurs en scène allaient ainsi pouvoir modifier le contenu du castelet virtuel et travailler ensemble à l'élaboration d'une scène sans avoir à être au même endroit.

1.4 Solutions disponibles

Il existe actuellement une panoplie d'applications de RV de toutes sortes. Les domaines les plus riches en la matière sont sans doute ceux du jeu vidéo et des simulateurs de vol. Les premières applications de RV ont d'ailleurs vu le jour dans ces domaines. Cependant, de nos jours, la RV a su s'implanter dans une foule de domaines. La médecine (planification de chirurgie, visualisation d'organes internes...), l'architecture (évaluation du design...) et la chimie (visualisation de molécules...) ne sont que trois exemples de domaines dans lesquels nous retrouvons maintenant des applications de RV. La RV est aussi grandement utilisée pour visualiser des données et pour faire des simulations de toutes sortes. Elle peut ainsi être utilisée en météorologie ou la quantité de données est très importante ou encore pour entraîner des futurs pilotes ou des futurs contrôleurs de centrales énergétiques. La RV peut même être utilisée pour former les gens à réagir en cas de feu.

Le domaine des Arts trouve aussi son compte dans la panoplie d'applications de RV disponibles de nos jours. Il est par exemple possible, grâce à des applications de RV conçues à cet effet, de faire la visite d'un musée d'art situé sur un autre continent sans avoir à s'y rendre physiquement. L'Art a aussi vu émerger des applications de RV plus axées sur la création. Cette technologie est par exemple utilisée pour composer de la musique ou encore comme nouveau support de création visuelle en remplacement des toiles traditionnellement utilisées par les peintres.

De nombreux projets mariant Art et RV ont déjà vu le jour. Le centre de recherche et de développement dans le domaine de la RV et du multimédia *Banff New Media Institute* [14] a d'ailleurs mis sur pied un projet nommé *Art and Virtual Environments* dans le but d'encourager cette union. Cependant, malgré l'abondance des applications de RV disponibles, aucune d'entre elles ne rencontrait tous les objectifs spécifiques déjà mentionnés à propos de l'application de RV désirée pour les fins du projet de Castelet Électronique du LANTISS. Ainsi, l'application de RV en question allait devoir être créée sur mesure pour répondre à ces besoins.

1.5 Solution proposée

Suite à l'analyse des solutions disponibles et du constat qu'aucune des applications de RV existantes ne répond à toutes les attentes et objectifs précis du projet de Castelet Électronique, notre projet de maîtrise est devenu celui de créer une application de RV sur mesure répondant à des besoins bien précis. L'application en question allait devoir être utilisable pour faire de la mise en scène. Cette caractéristique peut sembler évidente lorsque placée dans le contexte du projet de Castelet Électronique, mais allait distinguer cette application de bien des applications de RV existantes. En effet, l'application désirée allait devoir permettre à son utilisateur de créer et de modifier des mondes virtuels comme bon lui semble. Cela est différent des applications courantes qui proposent plutôt un monde déjà créé à leur utilisateur, lequel a la possibilité d'interagir avec celui-ci, mais souvent pas la possibilité de le modifier au point d'ajouter ou de supprimer des éléments, fonctionnalités pourtant spécifiquement recherchées dans l'application envisagée. D'autre part, l'application en question allait être utilisée par des metteurs en scène et non des experts en réalité virtuelle. Ainsi, l'application allait devoir être conviviale pour que des personnes souvent sans aucune formation en la matière puissent l'utiliser. Cette application était aussi particulière dans le sens qu'elle devrait permettre l'utilisation du castelet réel comme périphérique d'interaction avec le monde virtuel. Cette caractéristique n'est pas commune aux applications de RV et n'est d'ailleurs disponible dans aucune des applications dont les caractéristiques ont été éva-

luées. Une autre caractéristique que devait présenter l'application envisagée était celle de permettre la collaboration entre les metteurs en scène. Cette caractéristique allait être rendue possible par le développement d'une application fonctionnant en réseau et pouvant partager un monde virtuel entre plusieurs utilisateurs. Quoiqu'il ne s'agisse pas encore d'une caractéristique très répandue, celle-ci se retrouve tout de même déjà dans quelques applications de RV. Cela était tout de même un défi supplémentaire à relever dans l'implantation et le développement de l'application. Finalement, le dernier objectif spécifique à l'application envisagée était que cette dernière devait profiter des ressources disponibles à l'Université Laval, ce qui se traduisait dans ce cas par profiter des installations du LVSN [1] qui possède une salle de RV. Cette salle est entre autres munie d'un écran géant permettant une visualisation stéréoscopique (voir photo de la figure 1.9) dont il était évidemment intéressant de profiter.

Les objectifs précédemment mentionnés constituent les principaux objectifs spécifiques imposés par le projet de Castelet Électronique. Évidemment d'autres buts intrinsèques au fait qu'il s'agit d'une application de RV allaient aussi devoir être atteints. Outre ceux-ci, nous nous sommes, en tant que programmeur et développeur, fixé des objectifs supplémentaires non demandés par le projet de Castelet Électronique, mais que nous trouvions personnellement intéressants. Un de ces objectifs était que l'application à développer allait devoir être exécutée sur n'importe quelle plate-forme. Cet objectif a surtout été motivé par le fait que, bien que Windows [15] soit de loin le système d'exploitation le plus répandu, d'autres systèmes comme Linux [16] ou Mac OS X [17] sont aussi disponibles et prennent une place de plus en plus importante du marché, en particulier dans les universités. Un de ces systèmes alternatifs (en l'occurrence Linux) est d'ailleurs installé dans la salle de RV du LVSN. Cet objectif n'allait en fait que rendre l'application plus polyvalente sans vraiment en compliquer le développement. Un autre objectif personnel que nous nous sommes fixé avant même le développement de l'application était de faire en sorte que même si cette application devait, selon les objectifs du projet, profiter des installations du LVSN, celle-ci puisse tout de même être exécutée sur n'importe quel ordinateur et non seulement celui de la salle de RV. Cette application allait donc devoir être une application de RV utilisable dans une salle possédant tous les périphériques nécessaires à une expérience de RV, mais aussi sur n'importe quel ordinateur sans nécessiter quelque périphérique exotique que ce soit. L'utilisation de l'écran et des lunettes permettant une visualisation stéréoscopique ou encore l'utilisation du castelet réel comme périphérique d'entrée allait donc devoir être possible, mais non nécessaire à l'utilisation de l'application. Finalement, comme nous le verrons plus loin, un des outils choisis pour le développement de l'application allait nous donner l'idée de nous fixer un autre objectif, celui de faire de cette application une application à code ouvert.

FIG. 1.9 – Salle de réalité virtuelle du LVSN de l'Université Laval.

Tel que mentionné, les scènes virtuelles visualisées avec notre application de RV spécialisée seront *collées* au monde réel par l'utilisation du castelet réel. Ces scènes virtuelles seront donc des représentations virtuelles de scènes réelles. Celles-ci seront en effet constituées d'avatars, c'est-à-dire des objets virtuels représentant des objets (ou personnes) physiques réels. En ce sens, nous trouvions ce nom plutôt significatif et avons ainsi décidé d'appeler notre application **Avatar**.

1.6 Contenu du mémoire

Ce mémoire présente les travaux réalisés durant notre maîtrise et en particulier **Avatar**, le résultat de ces travaux. À cette fin, le chapitre 2 définit plus précisément les diverses caractéristiques qu'**Avatar** devait présenter en tant qu'application de RV ainsi que comment les objectifs prémentionnés concernant **Avatar** allaient pouvoir être atteints par cette application de RV. C'est aussi dans ce chapitre que sont présentées les fondations d'**Avatar**, c'est-à-dire les librairies que nous avons exploitées pour construire notre application de RV. Ce chapitre se termine par des résultats plus concrets, c'est-à-dire par la présentation de l'application de RV qui est à la base de ce qu'allait devenir **Avatar**.

Contrairement au chapitre 2 qui présente **Avatar** comme une application de RV générale, le chapitre 3 s'intéresse plutôt à une caractéristique spécifique à **Avatar**, à savoir celle de permettre l'utilisation d'acteurs réels comme périphériques d'entrée à cette application. On y expose premièrement différentes approches envisageables. La solution choisie est ensuite expliquée en détail. Ce troisième chapitre présente finalement certains résultats et expose certains problèmes liés à la solution choisie.

Tout comme le chapitre 3, le chapitre 4 se consacre à un objectif précis d'**Avatar**, c'est-à-dire celui de permettre la collaboration. Tel que mentionné précédemment, cet objectif n'est pas réellement unique dans le sens ou il existe déjà des applications de RV permettant le partage de monde virtuel. Cependant, dans le cadre spécifique d'**Avatar**, et en particulier dû à la possibilité qu'offre cette application d'utiliser des acteurs réels, cette caractéristique apporte des défis supplémentaires que les applications de RV standards n'ont pas à relever. Les principales difficultés sont donc exposées dans ce chapitre et une solution est ensuite suggérée. Des résultats viennent finalement appuyer la validité de la solution.

Le chapitre 5 conclut le mémoire en présentant les principales contributions apportées par nos travaux. Les principaux défis rencontrés durant ces travaux y sont

également exposés. La discussion se clôt finalement sur l'avenir d'**Avatar** et sur des suggestions d'améliorations envisageables pour cette application.

Chapitre 2

Avatar : une application de réalité virtuelle

2.1 Définition d'Avatar

Malgré toutes les exigences particulières mentionnées au chapitre précédent, **Avatar** allait d'abord et avant tout être une application de RV. Ainsi, le premier objectif à atteindre dans la réalisation de notre projet de maîtrise était d'élaborer et de développer une application de RV au sens large.

Or, selon Sherman et Craig [33], les quatre éléments clés de toute expérience virtuelle sont :

- Le monde virtuel ;
- L'immersion ;
- La rétroaction sensorielle ;
- L'interactivité.

Avatar allait donc devoir incorporer ces quatre éléments communs à toute application de RV. Cependant, **Avatar**, qui allait essentiellement être une application de RV, devait aussi respecter certains objectifs spécifiques dont il fallait tenir compte durant la phase de conception. Il prévalait en fait de faire en sorte que les quatre éléments clés essentiels à toute bonne expérience virtuelle fassent partie intégrante d'**Avatar** tout en respectant les objectifs plus précis propres à l'application désirée pour les fins du projet de Castelet Électronique.

2.1.1 Le monde virtuel

Un des principaux objectifs d'**Avatar** était d'être une application de RV utilisable pour faire de la mise en scène. Plus concrètement, **Avatar** allait donc devoir permettre à ses utilisateurs de créer des scènes virtuelles. Or ces scènes virtuelles sont en fait des mondes virtuels à part entière. Cet élément allait donc naturellement faire partie d'**Avatar**. Cependant, contrairement à la plupart des médiums incorporant des mondes virtuels (comme les livres, les films, les jeux vidéo et même comme bien des applications de RV), **Avatar** n'allait pas imposer de monde virtuel précis à ses utilisateurs. Au contraire, les mondes virtuels dans **Avatar** allaient plutôt être créés de toutes pièces, sur mesure et être entièrement paramétriques et modifiables par l'utilisateur. Le premier élément clé nécessaire à toute expérience virtuelle, à savoir le monde virtuel, n'allait donc pas seulement être qu'un élément constitutif d'**Avatar**, il devait en fait en être l'objet même.

Il est important de noter cependant que même si cette application allait permettre la création de mondes virtuels sur mesure, **Avatar** n'avait pas pour objectif de devenir une application de modélisation. Cela revient à dire que le rôle d'**Avatar** n'allait pas être de permettre de créer des acteurs virtuels, mais plutôt de permettre de visualiser des mondes virtuels créés sur mesure à partir d'acteurs déjà existants. Il existe une foule de logiciels de modélisation sur le marché ([7], [8], [9]) pouvant servir à créer de tels acteurs et, tel que mentionné précédemment, il est même possible de télécharger de tels acteurs de l'Internet.

2.1.2 L'immersion

Tel que mentionné précédemment, le LVSN de l'Université Laval où ont été réalisés nos travaux de maîtrise possède une salle de RV. Un des objectifs du projet de Castelet Électronique était d'ailleurs de profiter de ces installations. Cette salle est entre autres dotée d'un écran géant à rétroprojection. Lorsque utilisé en conjonction avec des lunettes à cristaux liquides *CrystalEyes* [18] aussi disponibles dans cette salle, ce système offre un affichage stéréoscopique permettant une visualisation en 3 dimensions, un élément important à l'immersion d'un utilisateur dans un monde virtuel. Grâce aux grandes dimensions de l'écran, l'utilisateur de ce système peut même, en plus de voir les éléments du monde virtuel en 3D, visualiser ceux-ci en grandeur réelle. Cette possibilité vient alors amplifier l'immersion de l'utilisateur dans le monde virtuel. Afin de pouvoir profiter des possibilités offertes par cette technologie et ainsi être immersive comme toute application de RV doit l'être, **Avatar** allait devoir offrir une sortie vidéo

stéréoscopique compatible avec les lunettes *CrystalEyes*.

Cependant, l'un de nos objectifs dans ce projet était de faire en sorte que **Avatar** puisse être utilisé sur n'importe quel ordinateur et non pas uniquement sur l'ordinateur de la salle de RV du LVSN. Les metteurs en scène qui le désireraient allaient évidemment pouvoir venir utiliser **Avatar** dans cette salle, mais l'application allait aussi pouvoir être utilisée n'importe où et en particulier sur des ordinateurs n'étant pas munis d'un système d'affichage stéréoscopique de type *CrystalEyes*. Ainsi, **Avatar** allait devoir aussi permettre une visualisation bidimensionnelle standard telle celle rencontrée chez les applications de modélisation et de rendu 3D. L'immersion allait évidemment être beaucoup moins importante dans ce cas, mais l'utilisation de l'application de RV allait du moins être tout de même possible.

D'autre part, l'utilisation de la technologie *CrystalEyes* n'est pas la seule façon d'obtenir une visualisation en 3D. Les anaglyphes [19] sont une autre façon de faire. C'est ce principe qui est à l'œuvre lorsque des images sont visualisées en 3D grâce à l'utilisation de lunettes constituées d'un verre rouge et d'un verre bleu comme celles fournies dans la couverture de ce mémoire.

Cette technologie est évidemment beaucoup plus simple et beaucoup moins dispendieuse que celle du type *CrystalEyes* et possède en plus l'avantage de pouvoir être utilisée sur n'importe quel ordinateur muni d'un écran couleur, ce qui va directement dans le sens de nos objectifs. Pour cette raison, **Avatar** allait aussi offrir, en plus des deux modes de visualisation déjà mentionnés, un mode d'affichage anaglyphique permettant l'utilisation de lunettes rouge et bleue afin de visualiser les mondes virtuels en 3D à peu de frais.

2.1.3 La rétroaction sensorielle

Il existe actuellement différents dispositifs spécialisés permettant d'obtenir avec une bonne précision la position et l'orientation d'un objet dans l'espace [20] [21]. Lorsqu'un tel dispositif est fixé à un individu (habituellement au niveau de la tête et parfois aussi au niveau des mains), un système de RV peut offrir une rétroaction sensorielle (la plupart du temps visuelle) à cet usager en fonction de sa position et de son orientation. De fait, avec un tel dispositif, l'utilisateur du système de RV se déplaçant dans le monde réel subit un déplacement correspondant dans le monde virtuel et obtient ainsi toujours un stimulus (visuel) cohérent avec ses actions.

La rétroaction sensorielle était sans aucun doute l'élément clé de l'expérience vir-

tuelle le plus difficile à offrir dans **Avatar**. Cela était encore plus difficile lorsque **Avatar** était envisagé du point de vue d'une application de RV devant pouvoir être exécutée sur n'importe quel ordinateur. Il fallait dans ce cas trouver un moyen de gérer la rétroaction sensorielle sans dispositif de positionnement, un périphérique hautement spécialisé dont ne disposent évidemment pas la plupart des ordinateurs. Cela allait de toute façon s'avérer nécessaire puisque la salle de RV du LVSN ne dispose d'aucun dispositif de positionnement.

Afin de remédier à ce problème, un système de navigation spatiale 3D allait devoir être développé sur les bases uniques d'une souris, un périphérique dont disposent actuellement tous les ordinateurs. Ce système de navigation n'allait évidemment pas avoir la prétention d'offrir les mêmes possibilités qu'un système de positionnement. Cependant, celui-ci allait tout de même offrir à l'usager la possibilité de se positionner là où il le souhaite dans le monde virtuel. Cette option ne devait par ailleurs pas annuler la possibilité d'utiliser **Avatar** avec un dispositif de positionnement plus évolué. **Avatar** allait donc devoir être codé de façon à ce que l'utilisation d'une vraie technologie de positionnement soit facilement envisageable advenant sa disponibilité.

2.1.4 L'interactivité

Avatar allait être une application de RV ayant pour but de permettre à ses utilisateurs de créer des mondes virtuels sur mesure. Dans cette optique, l'unique possibilité de navigation dans un monde virtuel immersif créé à partir d'acteurs choisis par l'usager n'était évidemment pas suffisante. **Avatar** devait en effet en plus offrir à ses usagers la possibilité d'interagir avec ces acteurs virtuels afin de pouvoir entre autres les positionner et les orienter à leur guise dans le monde virtuel.

Au cours des années, l'industrie de la RV a su mettre au point divers périphériques plus ou moins spécialisés qui pourraient servir à cette fin. La figure 2.1 présente un exemple de périphérique souvent utilisé dans les environnements de RV afin d'interagir avec les éléments constitutifs des mondes virtuels.

Tout comme les 3 autres éléments clé de l'expérience virtuelle, l'interactivité dans **Avatar** allait devoir être développée dans le but d'être aussi disponible sur tout ordinateur et non seulement sur ceux possédant des périphériques d'interactivité. Ainsi, comme dans le cas de la rétroaction sensorielle, l'interactivité dans **Avatar** allait être réalisée à l'aide de la souris. Cependant, toujours comme dans le cas de la rétroaction sensorielle, **Avatar** allait être conçue de façon à ce qu'il soit très facile de développer la composante nécessaire à l'utilisation d'autres types de périphérique d'interactivité

FIG. 2.1 – Gant de RV *Pinch Glove* de la compagnie *Fakespace Systems inc.* [22] permettant l'interactivité avec les éléments constitutifs d'un monde virtuel.

advenant leur disponibilité.

2.2 Les fondations d'Avatar

Le but et les principes de fonctionnement d'**Avatar** étant bien établis, le temps était maintenant venu d'implanter toutes ces idées dans une application concrète. Or, de nos jours, une multitude d'outils sont disponibles dans le domaine de la conception logicielle. La prochaine étape dans l'élaboration d'**Avatar** consistait donc à déterminer sur quelles fondations allait s'appuyer cette application de RV.

2.2.1 L'affichage graphique

Évidemment, un des aspects fondamentaux de cette future application était que celle-ci allait permettre de visualiser des mondes virtuels (et ce, possiblement en 3D). Il était donc tout à fait naturel d'examiner les outils pouvant être utilisés comme bases de l'affichage graphique d'**Avatar**.

Bien qu'il soit toujours possible de le faire, il est très rare qu'une application contrôle directement le périphérique d'affichage d'un ordinateur. Ceci est dû au fait qu'il existe des solutions beaucoup plus conviviales pour le programmeur d'applications logicielles. En fait, tel qu'illustré à la figure 2.2, le processus de rendu d'une application passe généralement par différentes couches matérielles et logicielles.

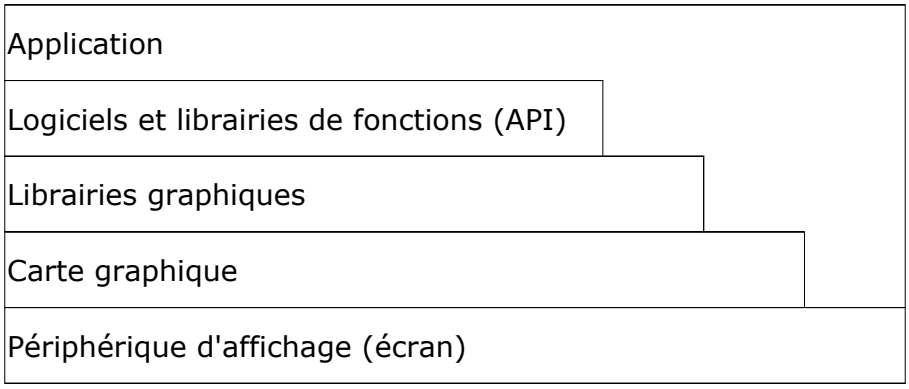


FIG. 2.2 – Hiérarchie des différentes couches matérielles et logicielles impliquées dans le processus de rendu graphique.

Bien qu'elles soient interreliées, ces couches sont toutes différentes et possèdent chacune leur utilité propre.

La carte graphique

La carte graphique est la couche matérielle responsable d'exécuter des commandes de haut niveau sur le périphérique d'affichage. Celle-ci s'occupe de gérer l'affichage pixel par pixel et met un éventail de primitives graphiques beaucoup plus conviviales à la disposition du programmeur. Ce dernier peut donc par exemple faire afficher un polygone sans avoir à se soucier de l'état de chacun des pixels un à un tel qu'il faut le faire lorsque l'on contrôle directement le périphérique d'affichage. Le passage par la couche matérielle qu'est la carte graphique apporte donc un avantage indéniable. Cependant, chaque carte graphique étant différente, un programme utilisant les commandes de

haut niveau disponibles avec une certaine carte graphique ne fonctionnera que sur les ordinateurs utilisant ce même matériel. C'est ainsi que prend tout son sens le passage par la couche logicielle formée par les librairies graphiques.

Les librairies graphiques

Les librairies graphiques jouent essentiellement le même rôle que les cartes graphiques, c'est-à-dire celui de gérer et d'exécuter des commandes de haut niveau formulées sur la base de primitives graphiques plus complexes que le simple pixel. Elles servent en fait d'interface aux cartes graphiques et font en sorte que les mêmes commandes puissent être utilisées sur une gamme plus étendue de machines. Ainsi, grâce à l'utilisation des librairies graphiques, le programmeur n'a plus à se soucier de quelle carte graphique sera utilisée, au même titre qu'il n'avait plus à se soucier de l'état de chaque pixel individuel de l'écran avec l'utilisation des cartes graphiques. Il est ainsi possible de bâtir une application qui pourrait être exécutée sur plusieurs types de machines en utilisant directement les librairies graphiques.

Les logiciels et les librairies de fonctions

Bien qu'elles soient très utiles et qu'elles permettent aux programmeurs de sauver beaucoup de travail, les librairies graphiques présentent aussi certaines limitations. En effet, une librairie graphique particulière interfacera tel ou tel type de matériel, mais n'interfacera pas nécessairement toutes les cartes graphiques disponibles. De plus, les commandes acceptées par les librairies graphiques, quoique moins élémentaires que celles acceptées par le périphérique d'affichage, sont généralement très limitées en complexité. Pour ces raisons, beaucoup d'applications tirent avantage de l'utilisation d'une couche logicielle de niveau encore supérieur aux librairies graphiques, à savoir celle constituée des logiciels et des librairies de fonctions servant d'interface de haut niveau aux différentes librairies graphiques. Certains de ces logiciels et librairies de fonctions peuvent même interfacer plusieurs librairies graphiques à la fois, permettant ainsi l'utilisation d'un seul ensemble de commandes de haut niveau afin d'effectuer des commandes graphiques très complexes sur un nombre encore plus grand de machines.

Solution choisie

Tel qu'illustré à la figure 2.2, les différentes couches matérielles et logicielles utilisables dans le processus de rendu graphique sont toutes directement accessibles à l'application et les couches supérieures s'appuient toujours sur les couches de plus bas niveau. Ainsi, les avantages issus d'une certaine couche sont toujours disponibles aux couches lui étant supérieures. De cette façon, l'utilisation de la couche de plus haut niveau est souvent suffisante.

Au moment d'implanter **Avatar**, plusieurs logiciels et librairies de fonctions pouvant servir d'interfaces de haut niveau aux librairies graphiques étaient disponibles, avec chacun leurs forces et leurs faiblesses respectives. De ce lot, une librairie de fonction nommée *The Visualization Toolkit* (VTK) [23] a particulièrement retenu notre attention pour différentes raisons.

Premièrement, VTK permet de gérer très facilement l'affichage stéréoscopique basé sur la technologie *CrystalEyes*, un élément très intéressant dans la perspective de l'objectif de profiter des installations du LVSN (de même que dans la perspective de toute application de RV).

Deuxièmement, VTK permet l'écriture de code C++ indépendant de la plate-forme sur laquelle il est compilé et exécuté. Cette caractéristique nous parut très intéressante du point de vue du programmeur. Celle-ci allait en effet nous permettre de coder **Avatar** de façon à ce qu'elle puisse être ensuite compilée sur différentes plates-formes sans aucune modification. Cette possibilité allait grandement simplifier la tâche d'atteindre notre objectif de faire d'**Avatar** une application pouvant être exécutée sur n'importe quelle plate-forme.

Finalement, bien que ce n'était pas absolument nécessaire et que cela ne faisait intervenir aucun des objectifs de départ, la dernière raison[1] ayant motivé le choix de VTK est qu'il s'agit d'un produit à code ouvert. Cette caractéristique nous a paru très intéressante et ce, à un point tel que nous avons par la suite décidé de donner le même statut à **Avatar**.

[1] Il existe différentes raisons pour lesquelles les diverses solutions évaluées pouvaient être mises de côté. Les caractéristiques exposées ici sont celles qui ont fait en sorte que VTK s'est démarqué parmi les solutions restantes.

2.2.2 L'interface utilisateur

Avatar allait donc utiliser les fonctionnalités offertes par VTK afin de présenter à l'utilisateur un rendu graphique des mondes virtuels qu'il allait créer. Cependant, afin de créer et de modifier ces mondes virtuels, l'usager allait devoir être en mesure d'interagir avec **Avatar**. En informatique, l'interaction avec une application ou un logiciel est réalisée par le biais de ce que l'on appelle une interface utilisateur. Ces interfaces peuvent prendre différentes formes. Afin de respecter l'objectif de convivialité qui allait permettre à des personnes n'ayant peut-être aucune formation en la matière d'opérer **Avatar** très facilement, l'application devait pour sa part disposer d'une interface utilisateur graphique (IUG).

C'est cette interface graphique qui allait permettre à l'utilisateur de faire exécuter différentes opérations à **Avatar**.

Tout comme pour l'affichage graphique, il existe certains outils informatiques permettant de grandement simplifier l'élaboration d'IUG. Parmi ceux-ci se trouve *Qt* [24], une librairie de fonctions C++ pour le développement de ce genre d'interfaces utilisateur. En plus d'être un outil très puissant permettant de sauver beaucoup de temps, *Qt* présente des caractéristiques allant directement dans le sens des objectifs de notre projet. En effet, tout comme VTK, *Qt* permet l'écriture d'un code unique portable sur les principales plates-formes sans aucune modification. *Qt* est aussi un produit à code ouvert pour quelques-unes de ces plates-formes. Cette librairie de fonctions allait ainsi être la deuxième fondation sur laquelle **Avatar** allait s'appuyer.

2.2.3 L'interface de programmation

Qt est un outil très puissant et très pratique permettant de créer des IUG pour toutes sortes d'applications. Or, lorsqu'il est utilisé pour gérer l'interface utilisateur d'une application, *Qt* prend le contrôle total de l'affichage graphique de cette application. Cependant, *Qt* n'est pas approprié pour faire du rendu 3D (en particulier du rendu stéréoscopique) et l'affichage graphique d'**Avatar** allait pour cette raison être géré par VTK. Ces deux outils qui allaient être utilisés comme fondations d'**Avatar** demandent tous deux le contrôle de l'affichage graphique et sont par le fait même fondamentalement incompatibles.

Cependant, l'interface de programmation d'application VTK_QT de Carsten Kübler [25] règle le conflit entre ces deux outils et permet l'utilisation simultanée de VTK

et *Qt*. Grâce à ce nouvel outil qu'est VTK_QT, *Qt* allait pouvoir être utilisé pour créer une IUG permettant d'interagir avec **Avatar** dont l'affichage graphique allait plutôt être géré par VTK.

Tout comme VTK et *Qt*, VTK_QT est un outil permettant l'écriture d'un code unique portable sur les principales plates-formes sans aucune modification et est aussi un produit à code ouvert.

Tous les outils qui allaient être utilisés pour faciliter le développement d'**Avatar** s'appuient sur le langage de programmation C++. Il était ainsi naturel de choisir ce langage de programmation pour coder l'application elle-même. D'ailleurs, lorsque les spécifications *ISO Standard C++* sont respectées, ce langage permet l'écriture d'un code portable sur toutes les plates-formes sans aucune modification.

2.3 Présentation d'Avatar, l'application de réalité virtuelle

Les outils présentés à la section précédente permettent de simplifier et d'accélérer le travail de programmation de la portion graphique d'une application. Néanmoins, la programmation d'une application n'est généralement pas chose simple et nécessite souvent beaucoup d'heures de travail. L'application désirée pour les fins de notre projet de maîtrise n'échappait pas à ce constat. Ainsi, plusieurs mois de travail furent nécessaires afin d'obtenir l'application de RV qu'est d'abord et avant tout **Avatar**.

Afin de présenter le résultat de ce travail et de voir cette application en action, nous présentons ici un exemple complet d'utilisation d'**Avatar**, de l'ouverture de l'application jusqu'à l'obtention du monde virtuel désiré. Comme un des objectifs de notre projet était qu'**Avatar** soit une application de RV pouvant servir à faire de la mise en scène, l'exemple présenté est celui de l'élaboration d'une scène de combat du film *The Matrix Reloaded* (2003) des frères Wachowski [26].

La figure 2.3 montre ce que l'utilisateur d'**Avatar** voit à l'ouverture de l'application. Tel qu'on peut le voir, celui-ci se retrouve en fait en présence d'une série de menus (lesquels sont des éléments de l'IUG par le biais de laquelle l'utilisateur peut interagir avec **Avatar**) et d'une fenêtre présentant un monde virtuel vide. Cette dernière peut être vue comme une fenêtre sur l'espace virtuel dans lequel seront placés les éléments constitutifs du futur monde virtuel. Tout monde virtuel existe nécessairement dans un

certain espace virtuel. Il s'agit donc là d'une structure élémentaire commune à tout monde virtuel et son imposition dès l'ouverture de l'application n'est alors aucunement restrictive pour l'utilisateur.

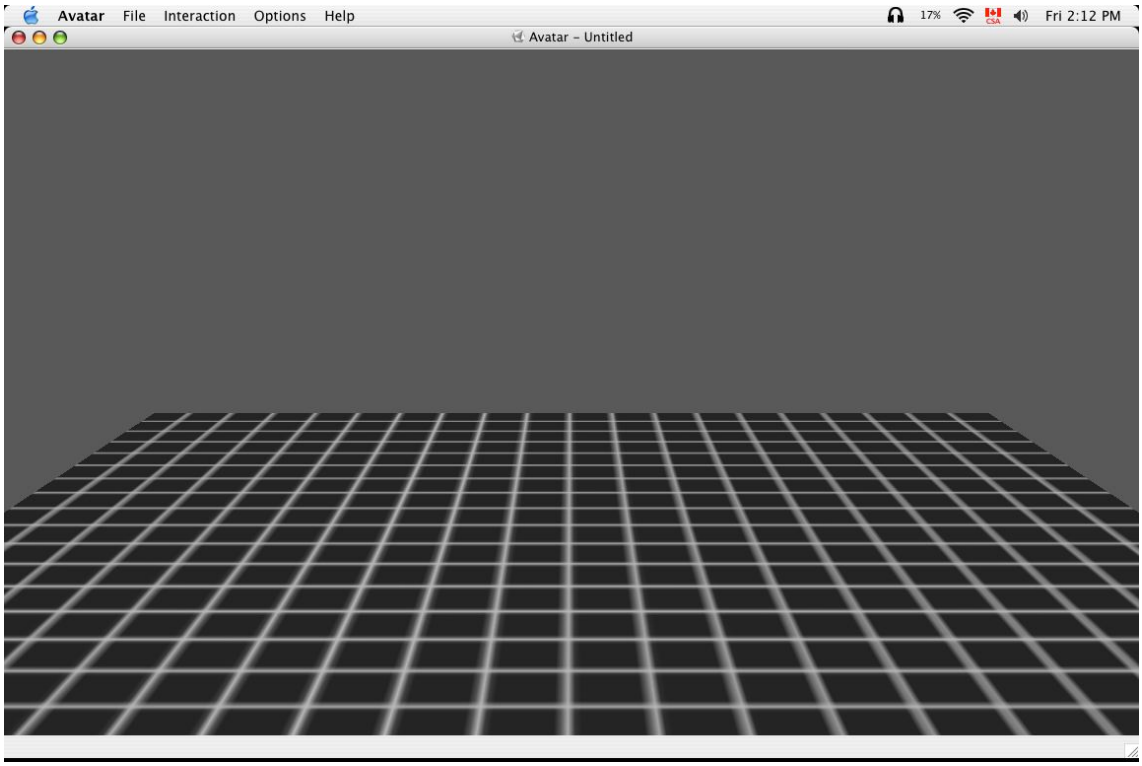


FIG. 2.3 – Monde virtuel vide présenté à l'utilisateur d'**Avatar** à l'ouverture de l'application.

Le lecteur notera assurément la présence d'une grille dans le monde virtuel prétendu vide présenté à la figure 2.3. Cette grille est présente dès l'ouverture d'**Avatar** et se retrouve dans tout monde virtuel visualisé avec cette application. Il ne s'agit cependant pas d'un constituant imposé à tout monde virtuel, mais plutôt d'une grille de référence aidant l'utilisateur à se repérer dans l'espace. Cette dernière illustre en fait une partie du plan $z = 0$.

À noter que l'utilisateur ne peut pas interagir avec cette grille de référence au même titre qu'il pourra le faire (rotation, translation, homothétie, etc.) avec les éléments qu'il ajoutera dans le monde virtuel. Cette grille ne fait donc pas partie des éléments constitutifs du monde virtuel à proprement parler.

Même si celle-ci est imposée à tout monde virtuel visualisé avec **Avatar**, l'utilisateur de notre application de RV a une certaine liberté face à cette grille de référence. En effet, tel qu'illustré à la figure 2.4, le menu *Options* d'**Avatar** permet entre autres d'afficher ou non cette grille, laquelle est visible par défaut.

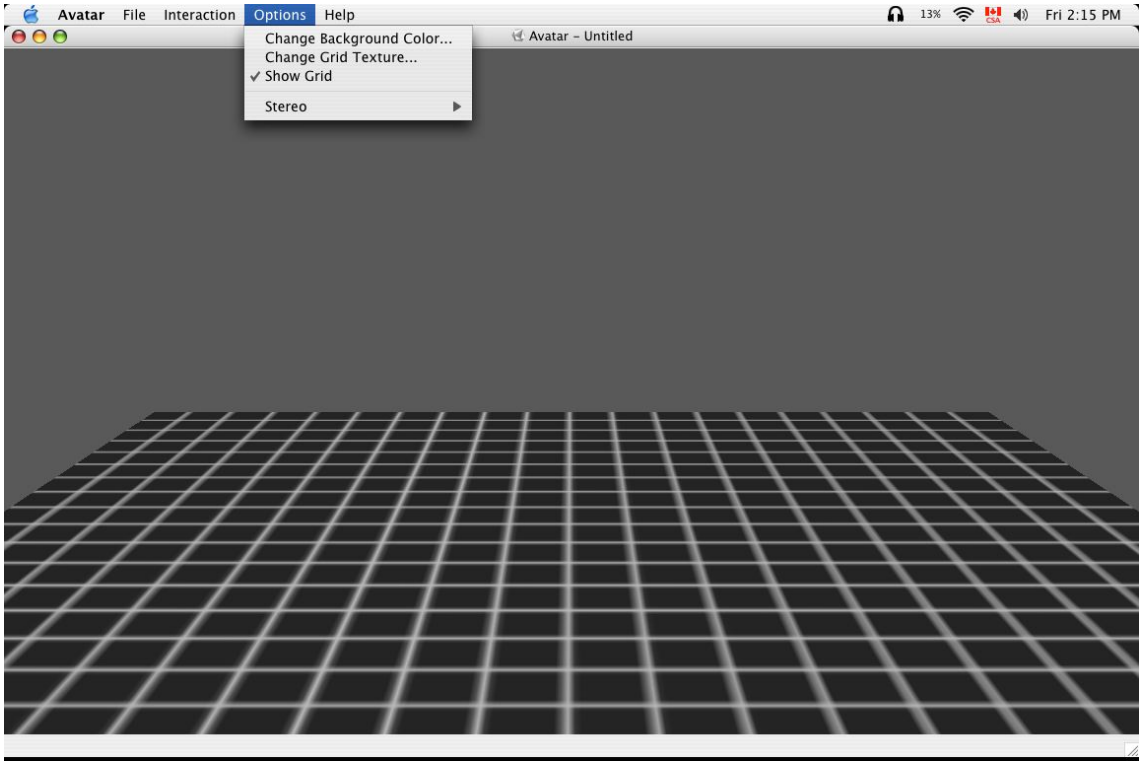


FIG. 2.4 – Le menu *Options* d'**Avatar**.

Tel qu'on peut également le voir sur cette même figure, le menu *Options* permet aussi à l'utilisateur de changer la texture de la grille de référence. Cette possibilité peut à première vue sembler plutôt banale, mais celle-ci se révèle au contraire très pratique lorsque vient le temps de concevoir un monde virtuel. En effet, comme cette grille est déjà positionnée en $z = 0$, il est possible (et pratique) de s'en servir comme plancher dans le monde virtuel.

La scène que nous désirons concevoir pour notre exemple se déroule à l'extérieur, sur un plancher constitué de tuiles de béton. La figure 2.5 illustre que ce plancher peut facilement être obtenu en appliquant simplement la texture appropriée à la grille de référence. Le lecteur remarquera également sur cette figure que la couleur de l'arrière-plan du monde virtuel a aussi été changée afin d'illustrer une des autres options disponibles dans **Avatar**.

La scène que nous avons choisie pour notre exemple présente des personnages en train de se battre. La prochaine étape du processus de mise en scène à l'aide d'**Avatar** consiste donc à ajouter ces personnages au monde virtuel. Cette tâche est réalisée par le menu *File* d'**Avatar**, lequel est présenté à la figure 2.6.

Cette figure illustre d'ailleurs d'autres fonctionnalités importantes d'**Avatar**. Nous

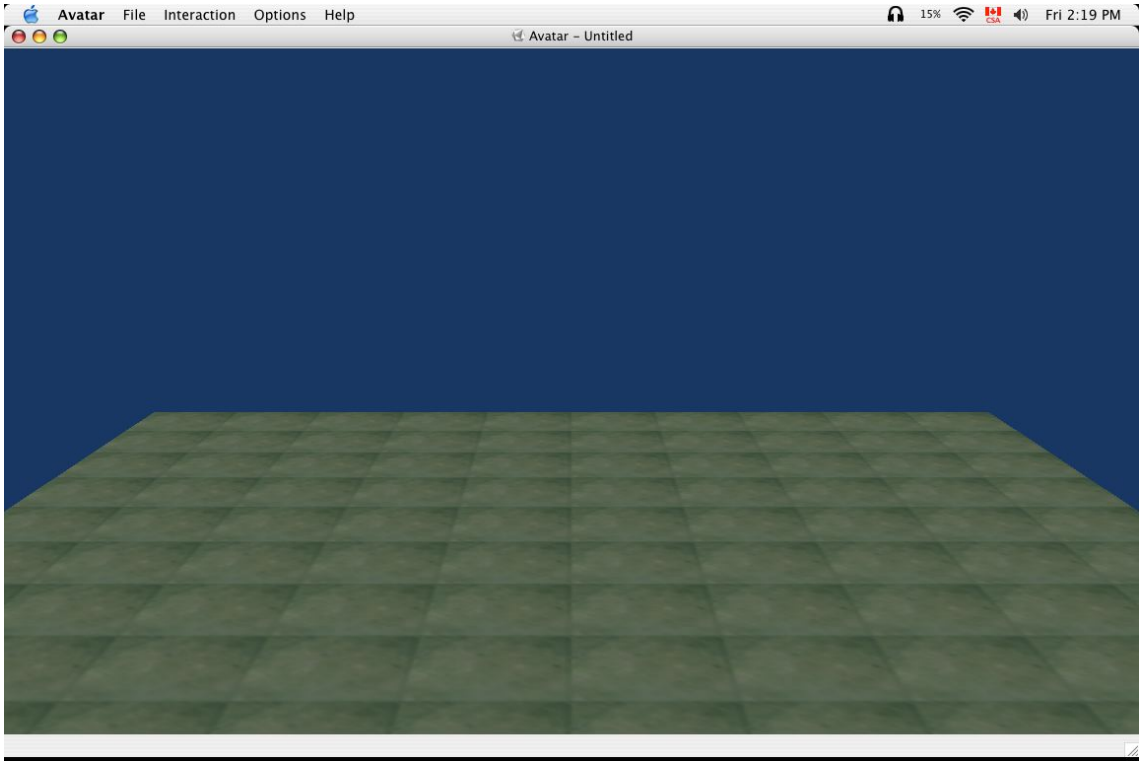


FIG. 2.5 – L'utilisateur d'**Avatar** a la possibilité de changer la couleur de l'arrière-plan ainsi que la texture de la grille de référence du monde virtuel.

pouvons en effet y observer qu'en plus de permettre d'ajouter des acteurs à la scène, le menu *File* permet entre autres à l'utilisateur de créer un nouveau monde virtuel (vide), d'ouvrir un monde virtuel préalablement sauvegardé et à l'inverse, de sauvegarder le monde virtuel courant.

La figure 2.6 illustre aussi que s'il ajoute un acteur au monde virtuel, l'utilisateur a alors la possibilité de choisir l'une des deux primitives d'acteurs préalablement définies dans **Avatar** (à savoir un cube ou une sphère) ou encore d'ajouter un acteur plus spécifique (non connu d'**Avatar**).

Dans ce dernier cas, l'utilisateur se voit alors présenter une fenêtre de sélection lui permettant d'indiquer à **Avatar** quel fichier contient l'information concernant cet acteur. L'utilisateur a alors en fait le choix de sélectionner des fichiers d'extension *3ds*, *wrl*, ou encore *tiff*. Les extensions *3ds* et *wrl* sont associées à deux formats de fichiers standards et très répandus au niveau des modèles 3D. Pour sa part, l'information contenue dans un fichier du format *tiff* ne décrit pas un modèle 3D, mais plutôt une image. Si un tel fichier est sélectionné par l'utilisateur d'**Avatar** pour être ajouté au monde virtuel, l'image contenue dans ce fichier sera alors appliquée sur un plan et c'est plutôt ce plan qui sera ajouté au monde virtuel. Les dimensions de ce plan sont

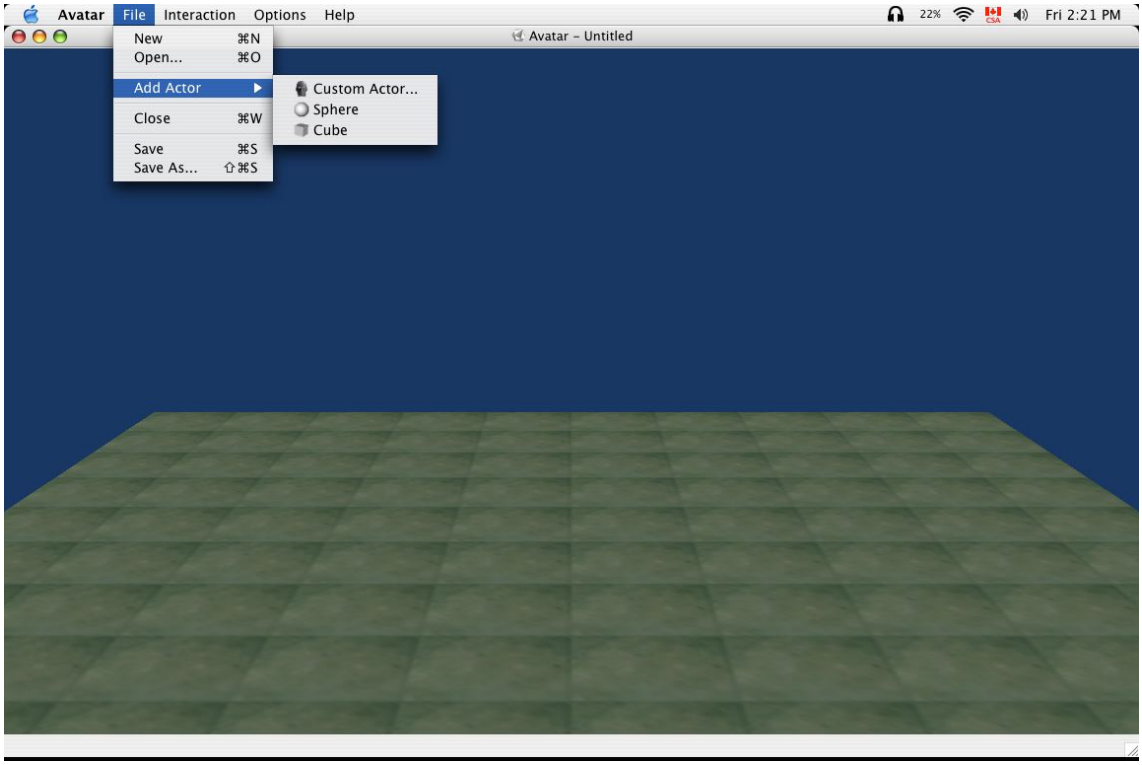


FIG. 2.6 – Le menu *File* d'**Avatar**.

pré-définies dans **Avatar**. Cependant, comme c'est le cas pour tout acteur ajouté dans **Avatar**, l'utilisateur a ensuite la possibilité d'appliquer une homothétie sur cet acteur (le plan) afin de le re-dimensionner pour qu'il respecte l'échelle du monde virtuel[2].

La figure 2.7 présente le résultat de l'ajout d'un premier acteur à notre scène de combat virtuelle. Lorsque l'on ajoute un acteur à un monde virtuel dans **Avatar**, celui-ci est par défaut toujours placé à l'origine. L'utilisateur a cependant ensuite la possibilité de positionner cet acteur à l'endroit désiré dans le monde virtuel. Ce positionnement est réalisé à l'aide de la souris. En fait, toute interaction avec un acteur d'un monde virtuel (rotation, translation, homothétie, repositionnement à l'origine, suppression) est réalisée à l'aide de la souris dans **Avatar**. Ainsi, bien qu'il soit possible de se servir d'**Avatar** avec une souris ne présentant qu'un seul bouton, il est préférable d'utiliser cette application avec une souris à 3 boutons[3]. La combinaison de ces différents boutons avec des touches spécifiques du clavier permet à l'utilisateur d'effectuer les diverses opérations possibles sur les acteurs du monde virtuel. L'aide d'**Avatar** (disponible par le menu *Help*) décrit en détail les différentes combinaisons disponibles ainsi que leur

[2]L'échelle du monde virtuel n'est pas gérée par **Avatar**. Cette gestion est laissée à l'utilisateur.

[3]Lorsque la souris utilisée avec **Avatar** présente moins de 3 boutons, les interactions associées aux boutons manquants peuvent être réalisées en appuyant sur les touches appropriées du clavier. Cette possibilité est explicitée dans l'aide d'**Avatar** (menu *Help*).

effet sur les acteurs virtuels.

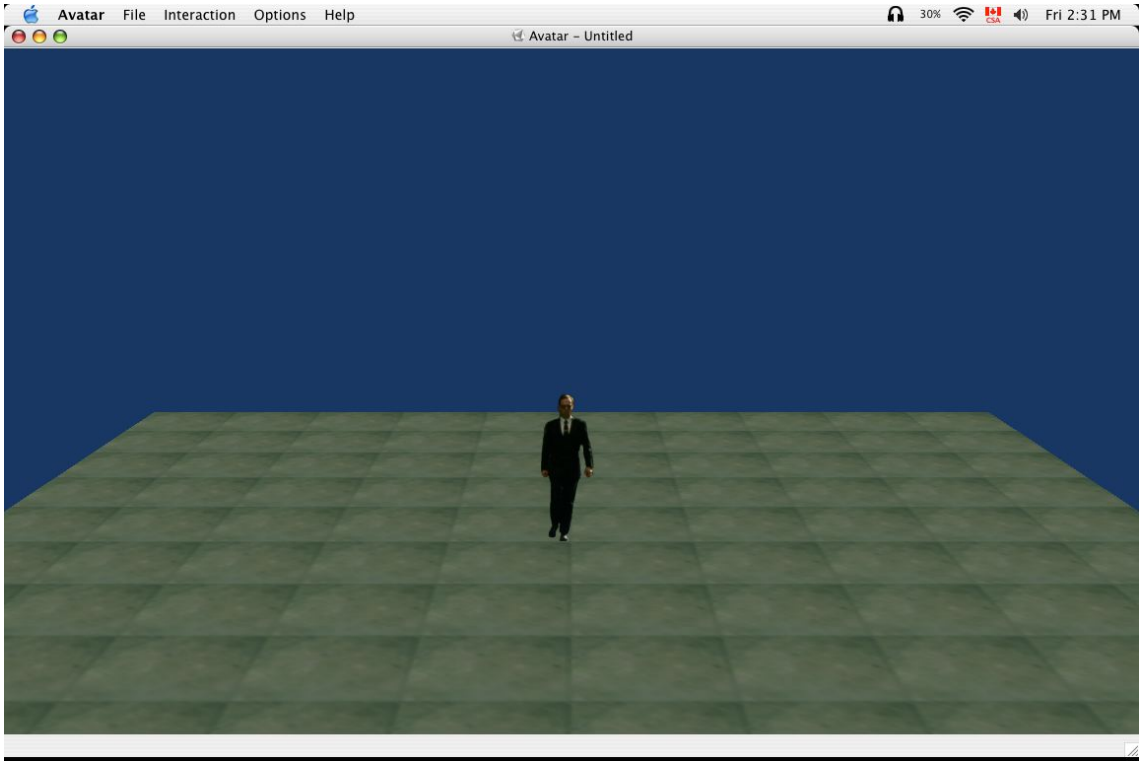


FIG. 2.7 – Lorsqu'un acteur est ajouté au monde virtuel, il est par défaut positionné à l'origine.

Il est pratique, lorsque l'on veut positionner des acteurs dans le monde virtuel, de pouvoir observer ce monde de différents point de vue. À cet effet, **Avatar** permet à son utilisateur de se positionner n'importe où dans le monde virtuel. Tel que mentionné précédemment, c'est aussi par le biais de la souris que l'utilisateur d'**Avatar** navigue dans le monde virtuel. Toujours en fonction des différentes combinaisons réalisées avec certaines touches spécifiques du clavier, la souris permet à l'utilisateur de réaliser différentes actions (rotation, pan, zoom, etc.). Encore une fois, ces différentes options sont détaillées dans l'aide d'**Avatar**.

La souris joue donc deux rôles distincts dans **Avatar**. En effet, elle permet d'une part à l'utilisateur d'interagir avec les acteurs virtuels et, d'autre part, de se déplacer dans le monde virtuel. Le rôle que joue la souris est en fait fonction du *mode* dans lequel se trouve **Avatar**, à savoir le mode *Actors* (utilisé pour interagir avec les acteurs) ou le mode *Camera* (utilisé pour naviguer dans le monde virtuel). Le passage d'un mode

à l'autre est réalisé par le menu *Interaction*[4].

En plus de lui permettre de se positionner n'importe où dans le monde virtuel, **Avatar** permet à son utilisateur d'enregistrer les points de vue qu'il trouve intéressants afin de pouvoir y revenir instantanément par la suite. Certains points de vue plus courants sont d'ailleurs pré-enregistrés et toujours disponibles dans **Avatar**.

Ces différentes possibilités offertes par **Avatar** sont toutes accessibles via le menu *Interaction*, lequel est présenté à la figure 2.8.

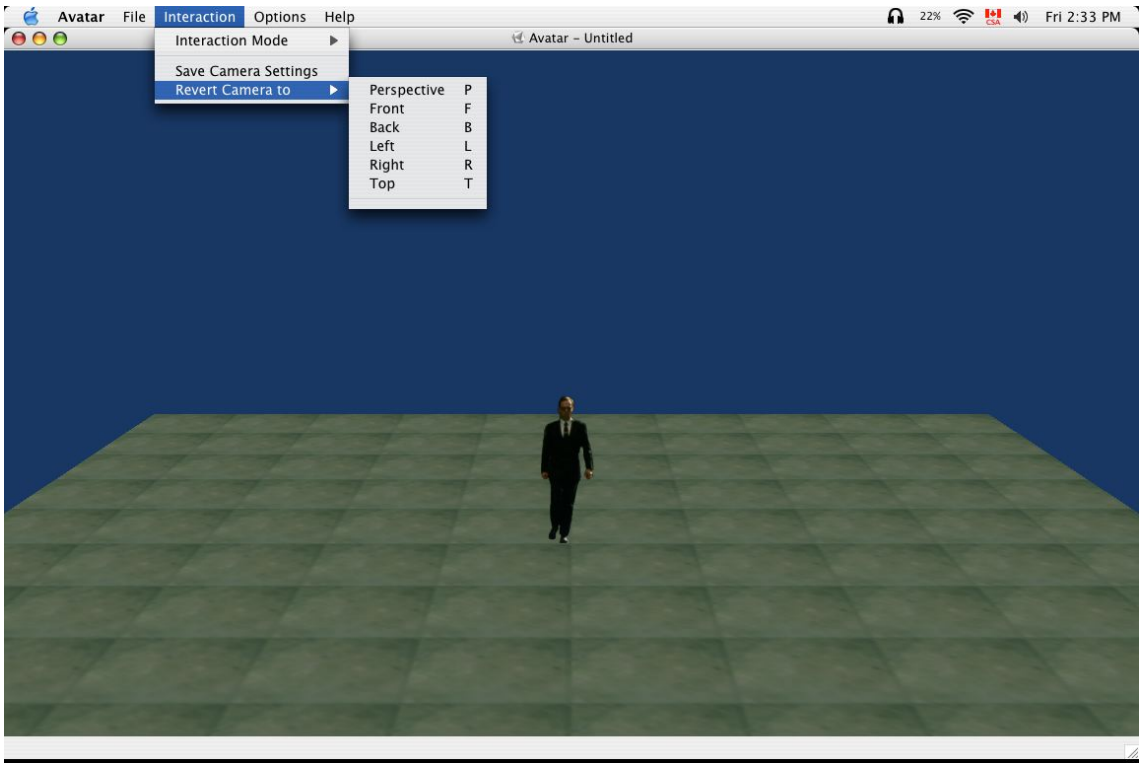


FIG. 2.8 – Le menu *Interaction* d'**Avatar**.

Les points de vue pré-enregistrés (*Perspective*, *Front*, *Back*, *Left*, *Right* et *Top*) sont souvent très pratiques pour positionner les acteurs dans le monde virtuel. L'acteur présent dans notre scène est d'ailleurs positionné en utilisant un de ces points de vue pré-enregistrés. En effet, la vue de dessus (*Top*) est idéale pour reculer l'acteur à la position désirée tel que l'illustre la figure 2.9.

La scène que nous désirons concevoir pour notre exemple fait intervenir plusieurs

[4]Des raccourcis clavier ont également été programmés afin de permettre à l'utilisateur d'**Avatar** de passer plus rapidement d'un mode à l'autre en appuyant simplement sur la touche 'A' pour le mode *Actors* ou la touche 'C' pour le mode *Camera*. À noter que la plupart des éléments présents dans les divers menus d'**Avatar** sont aussi accessibles par de tels raccourcis clavier.

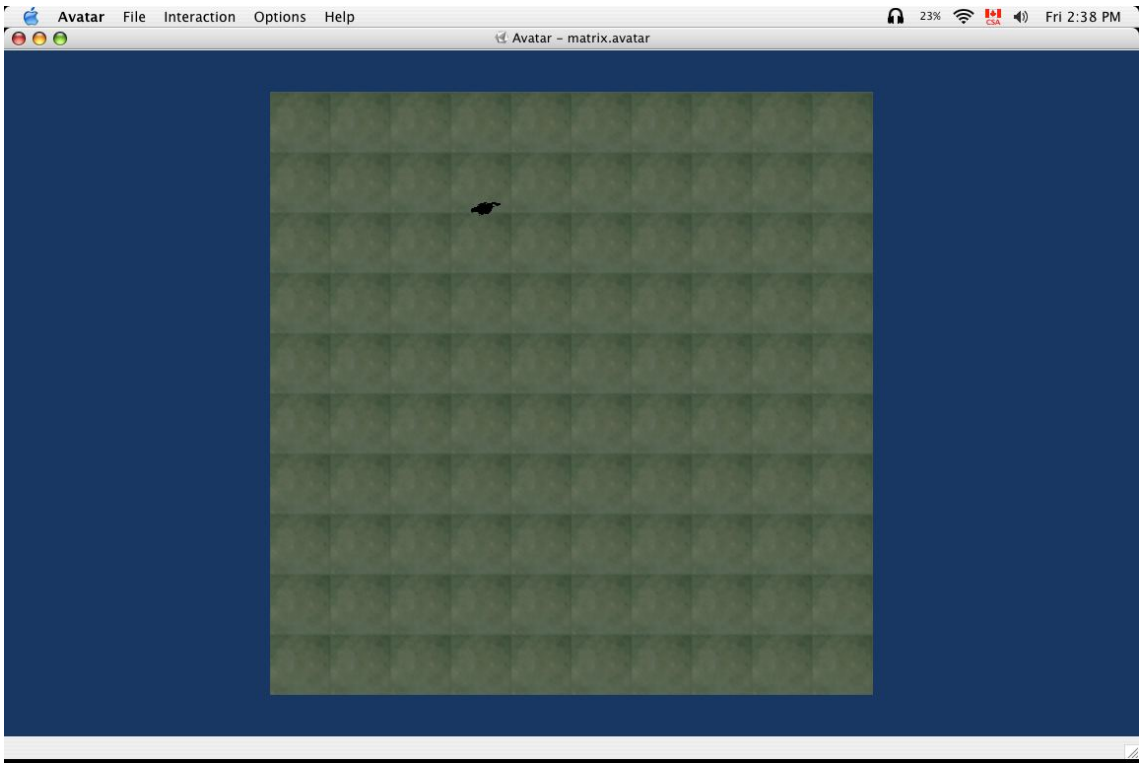


FIG. 2.9 – Des points de vue pré-enregistrés dans **Avatar** facilitent le positionnement des acteurs dans le monde virtuel.

acteurs. Chacun de ceux-ci est donc ajouté exactement de la même manière au monde virtuel, puis positionné à l'endroit désiré. La figure 2.10 présente cette scène une fois tous les acteurs ajoutés et positionnés.

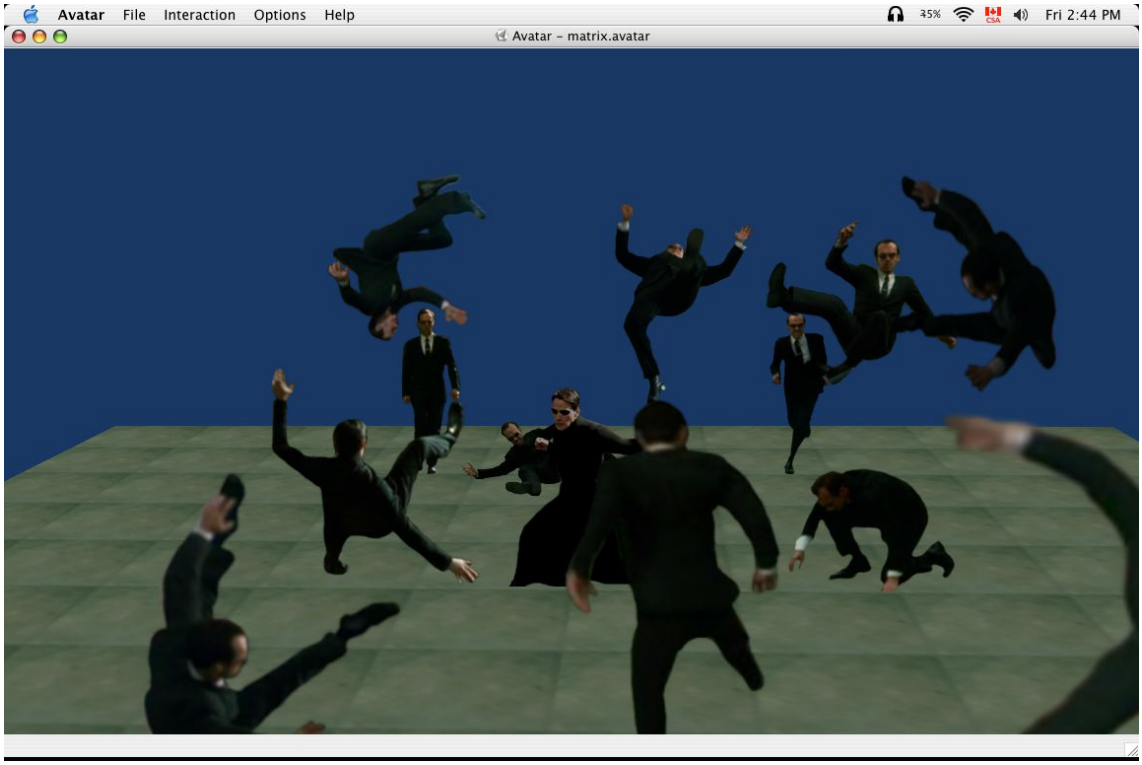


FIG. 2.10 – Exemple de monde virtuel impliquant plusieurs acteurs (monde virtuel élaboré avec **Avatar**).

L'action de notre scène se déroule à l'extérieur, dans une cour entre des bâtiments. Afin d'illustrer ce fait, nous pourrions ajouter tous les modèles 3D de bâtiments nécessaires à cette scène dans notre monde virtuel. Or, comme toute l'action se déroule au même endroit et que ces bâtiments sont assez loin dans l'arrière-plan, il est aussi possible d'utiliser simplement une image plane de ces bâtiments. De cette façon, aucune modélisation n'est nécessaire, une tâche qui peut parfois être ardue et fastidieuse. C'est donc cette option qui a été choisie pour notre exemple.

Les figures 2.11 et 2.12 présentent deux points de vue légèrement différents de la scène une fois le décor ajouté.

Il est souvent difficile de réaliser la profondeur d'une scène par le biais d'une image. C'est d'ailleurs pour aider le lecteur à cette tâche que deux points de vue différents de la scène ont été présentés. Or, même avec deux images, l'allure tridimensionnelle de la scène n'est pas entièrement rendue. Cette constatation nous amène à discuter d'une des options encore non abordée d'**Avatar**, à savoir celle de l'affichage stéréoscopique.

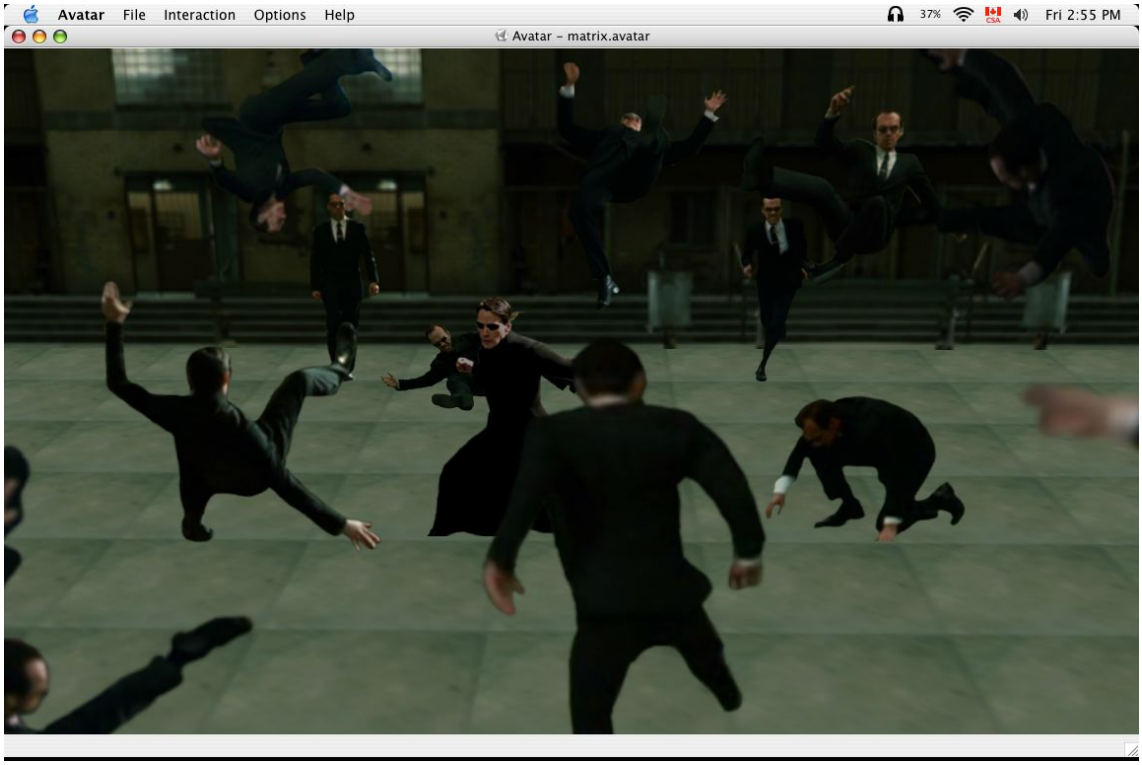


FIG. 2.11 – Exemple de monde virtuel élaboré dans **Avatar** afin de faire la mise en scène d'un film.

Avatar supporte en fait 3 modes d'affichage. Dans un premier temps, le monde virtuel peut être visualisé en 2 dimensions à l'aide d'un écran standard. Les images présentées jusqu'ici étaient d'ailleurs le résultat de captures d'écran où **Avatar** était utilisé avec ce mode d'affichage. Ce mode d'affichage ne nécessite aucun équipement spécialisé et concorde parfaitement avec notre objectif de faire en sorte qu'**Avatar** puisse être utilisé sur pratiquement n'importe quel ordinateur.

En plus de ce mode d'affichage *traditionnel*, **Avatar** supporte deux modes d'affichage stéréoscopique permettant une visualisation en 3 dimensions. Premièrement, afin de combler l'objectif du projet de Castelet Électronique de profiter des installations du LVSN, **Avatar** offre un affichage stéréoscopique compatible avec la technologie *CrystalEyes*.

Deuxièmement, afin de permettre une visualisation en 3 dimensions, mais toujours dans l'esprit de faire d'**Avatar** une application de RV utilisable sur n'importe quel ordinateur, un mode d'affichage anaglyphique (permettant la visualisation en 3D avec les lunettes rouge et bleue) est aussi disponible. Afin d'illustrer le potentiel de cette technologie et donner une idée au lecteur du genre de résultats qu'il est possible d'obtenir avec **Avatar** sur pratiquement n'importe quel ordinateur, la figure 2.13 présente une image

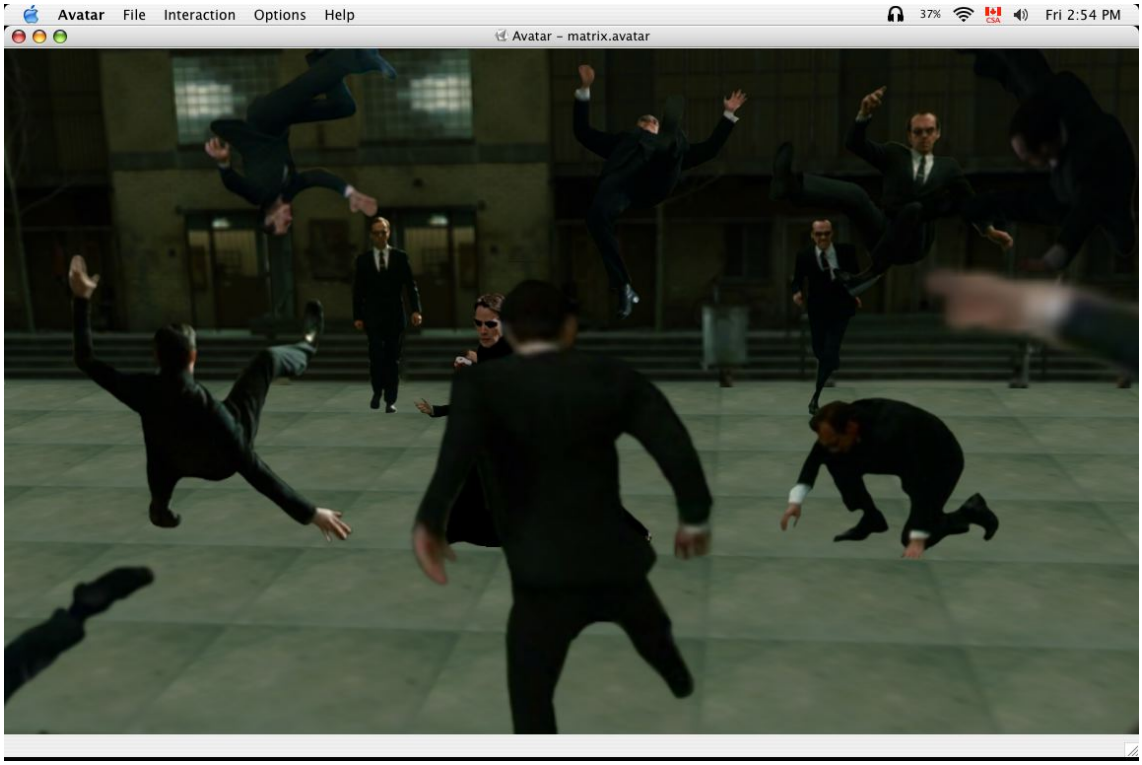


FIG. 2.12 – Point de vue légèrement différent du monde virtuel présenté à la figure 2.11.

de notre scène de combat lorsque ce mode d'affichage est utilisé. Le lecteur est donc maintenant invité à utiliser la paire de lunettes fournie dans la couverture intérieure de ce mémoire afin de visualiser notre scène en 3 dimensions.

2.4 Synthèse récapitulative

Cette première version d'**Avatar** a atteint plusieurs objectifs énoncés lors de la définition de notre projet. En premier lieu, cette application de RV est utilisable pour faire de la mise en scène. En effet, **Avatar** est une application de RV n'imposant aucun monde virtuel à son utilisateur, mais lui offrant plutôt la possibilité de créer des mondes virtuels de toutes pièces et sur mesure. L'utilisateur a d'ailleurs la possibilité de modifier le contenu de ces mondes virtuels durant l'utilisation même de l'application.

De par son IUG, **Avatar** est aussi une application très conviviale, ce qui était un autre objectif à atteindre étant donné le fait que cette application de RV allait possiblement être utilisée par des personnes n'ayant aucun bagage dans ce domaine.

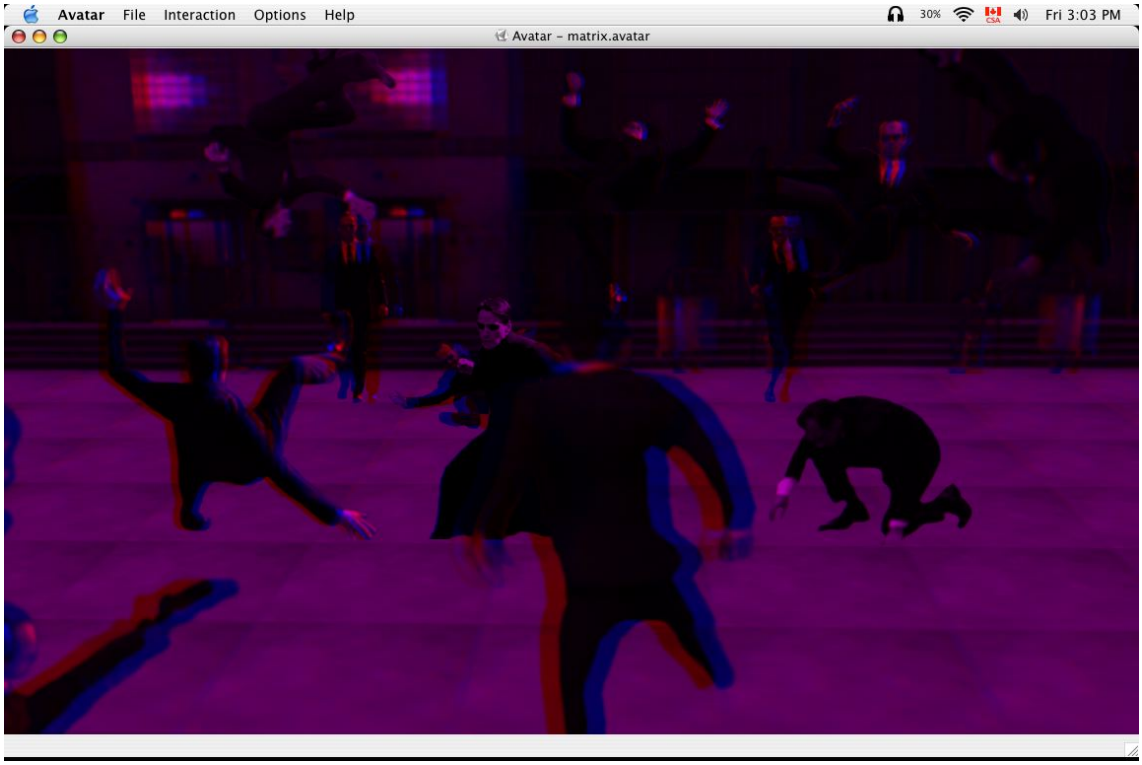


FIG. 2.13 – Rendu anaglyphique de la scène finale. **Avatar** permet 3 modes d'affichage dont 2 offrant des rendus stéréoscopiques.

Avatar a aussi été conçu en ayant en tête qu'il devait s'agir d'une application de RV pouvant profiter des installations disponibles au LVSN, mais devant tout de même pouvoir être exécutée sur n'importe quel ordinateur. Ces deux objectifs ont aussi été atteints, entre autres en offrant deux types d'affichage stéréoscopique en plus d'un affichage plus traditionnel.

Grâce à l'utilisation d'outils de programmation appropriés, le code d'**Avatar** peut être compilé sur n'importe quelle plate-forme et ce, sans aucune modification. Cette caractéristique permet de faire d'**Avatar** une application multi-plates-formes. **Avatar** est d'ailleurs actuellement disponible pour Linux, pour Mac OS X ainsi que pour Windows.

Finalement, **Avatar** est une application à code ouvert. Ainsi, autant le code que les différentes versions compilées de l'application (pour les différentes plates-formes) peuvent être téléchargées d'Internet. **Avatar** est en effet hébergé sur le site Internet de SourceForge [27] et est accessible via l'adresse `http://sourceforge.net/projects/avatar`.

Bien que cette première version d'**Avatar** ait atteint plusieurs des objectifs de notre projet de maîtrise, l'un des objectifs importants du projet de Castelet Électronique n'a

pas encore été abordé : celui de permettre l'utilisation du castelet réel afin de contrôler le castelet virtuel. Cet objectif constitue à lui seul un défi de taille et est en fait l'objet du chapitre suivant.

Chapitre 3

Utilisation d'acteurs réels pour interagir avec le monde virtuel

3.1 Définition du problème

Selon sa définition, **Avatar** se devait d'être d'abord et avant tout une application de RV. Cependant, cette application de RV se devait aussi de respecter certains objectifs très précis. Après tout, à la base, le projet qui a mené au développement d'**Avatar** avait pour objectif principal le développement d'une application de RV pouvant être utilisée comme nouvel outil de mise en scène. En fait, **Avatar** a été développé dans le but d'être l'application de RV qui allait permettre de visualiser le castelet virtuel du projet de Castelet Électronique. **Avatar** allait donc entre autres être utilisé par des metteurs en scène, lesquels sont habitués à travailler avec des maquettes. Or l'interactivité offerte par le biais de la souris dans **Avatar** est beaucoup moins intuitive et directe que celle offerte par les maquettes. Celle-ci se rapproche en effet grandement de l'interactivité disponible dans les logiciels de modélisation et de rendu 3D, lesquels présentent une lacune à ce niveau. De plus, tel que mentionné précédemment, l'introduction du castelet virtuel dans le projet de Castelet Électronique n'avait pas pour but de priver les metteurs en scène d'un outil qu'ils apprécient (en l'occurrence la maquette), mais plutôt d'augmenter les possibilités et fonctionnalités de celui-ci. Ainsi, l'un des objectifs de ce projet était de permettre l'utilisation du castelet réel du LANTISS (une maquette générique, motorisée et munie d'un éclairage à base de fibres optiques) afin d'interagir avec le castelet virtuel (visualisé à partir d'**Avatar**).

Cet objectif est évidemment spécifique au projet de Castelet Électronique. Or, **Ava-**

tar, qui peut très bien servir à visualiser le castelet virtuel dans ce projet, est cependant une application de RV beaucoup plus générale pouvant être utilisée à bien d'autres égards. Néanmoins, **Avatar** se devait de respecter tous les objectifs du projet de Castelet Électronique étant donné que cette application de RV générale a en premier lieu été développée dans le but d'être utilisée pour ce projet spécifique. Ainsi, afin d'être à la fois une application de RV générale pouvant être utilisée par n'importe qui, sur n'importe quel ordinateur, dans n'importe quel but et du même coup respecter tous les objectifs du projet par lequel cette application était née, il fallait traduire cet objectif spécifique du projet de Castelet Électronique en un objectif plus générique de l'application de RV générale qu'est **Avatar**. En ce sens, cet objectif spécifique devenait l'objectif générique de permettre l'utilisation d'acteurs réels comme périphériques d'interaction avec **Avatar**. Ces nouveaux périphériques allaient en fait devoir jouer le rôle habituellement interprété par un gant de RV dans les applications de RV plus standards (voir à ce sujet la section 2.1.4).

L'objectif était donc de taille. Il fallait être en mesure d'associer des acteurs réels aux acteurs du monde virtuel et faire en sorte que l'interaction avec un acteur réel se répercute exactement de la même manière sur l'acteur virtuel correspondant. Il fallait donc trouver un moyen d'obtenir la position et l'orientation 3D d'acteurs réels puis de transmettre celles-ci à **Avatar** afin qu'il puisse affecter la même configuration aux avatars (les acteurs du monde virtuel correspondants aux acteurs réels).

La solution qui fut proposée au LANTISS était d'utiliser une caméra observant la scène réelle et de calculer la pose[1] 3D des acteurs de cette scène réelle à partir des images obtenues de la caméra. Cette pose allait ensuite être transmise à **Avatar** afin qu'il gère le castelet virtuel en conséquence. Étant donné que les webcams sont de nos jours très abordables et très répandues, cette solution envisagée concordait bien avec notre objectif de faire d'**Avatar** une application de RV pouvant être utilisée sur n'importe quel ordinateur et non seulement dans une salle de RV munie d'équipements hautement spécialisés.

Il existe différentes façons d'estimer la pose 3D d'un objet à partir d'images 2D ; une revue de littérature allait nous permettre de choisir la manière la plus appropriée à notre situation.

[1] En vision par ordinateur, le terme *pose* désigne l'information de position et d'orientation spatiale d'une composante dans l'espace cartésien.

3.2 Revue de littérature

Dans le domaine de la vision par ordinateur, il existe un champ d'étude appelé *réalité augmentée* (RA) que Milgram situe entre l'environnement réel et l'environnement virtuel dans son continuum réalité-virtualité [47]. Contrairement à ce que l'on retrouve en RV où tout l'environnement (virtuel) est généré par ordinateur, en RA, les objets virtuels sont plutôt ajoutés à l'environnement réel. Cet ajout est la plupart du temps réalisé sur des images de l'environnement réel, lesquelles sont ensuite présentées à l'utilisateur du système de RA. Ce concept trouve de plus en plus d'applications de nos jours et particulièrement dans le monde de la télévision et dans celui du cinéma. La figure 3.1 présente d'ailleurs deux images illustrant des applications de la RA.

Afin que les objets virtuels ajoutés par les systèmes de RA semblent faire réellement partie de l'environnement réel, de tels systèmes doivent être en mesure d'estimer la géométrie de la scène observée par la caméra. Ces systèmes doivent donc, en plus de connaître la géométrie des objets du monde réel (les acteurs réels), déterminer précisément la relation existant entre ces objets et la position de la caméra. Ceci revient à dire que les systèmes de RA doivent être capables d'estimer la pose 3D des acteurs réels relativement à la caméra les observant. C'est cette information qui permet ensuite aux systèmes de RA d'ajouter les objets virtuels de façon cohérente aux images du monde réel.

Beaucoup de travail a donc été réalisé sur le problème d'estimation de pose au cours des dernières années dans le champ de la RA [48] [49]. Ces travaux ont d'ailleurs mené à l'émergence de plusieurs méthodes d'estimation de la pose 3D d'objets relativement à la caméra les observant. Dans la grande diversité des méthodes développées, plusieurs fonctionnent en mettant en correspondance des modèles connus des objets d'intérêts (les objets dont on veut estimer la pose) avec les images capturées par la caméra. Ce sont ces méthodes qui ont retenu notre attention.

Certaines de ces méthodes [45] [46] utilisent une approche que l'on pourrait qualifier de globale (par opposition à locale) en ce qu'elle travaille sur l'objet d'intérêt dans son ensemble. Les méthodes basées sur cette approche consistent à comparer l'image capturée par la caméra à une multitude d'images d'entraînement (le modèle connu) de l'objet d'intérêt afin de déterminer si cet objet est présent dans l'image de la caméra. Ce type de méthode est surtout utilisé à des fins de détection automatique d'objets d'intérêt dans une image, mais peut aussi être utilisé pour estimer la pose 3D de ces objets (dans ce cas la pose 3D de ces objets dans les images d'entraînement doit évidemment être connue). Les résultats de l'estimation de la pose obtenus avec ces méthodes ne

(a) Depuis 1998, le système de RA *1st & Ten* de Sportvision [31] permet aux amateurs de football de visualiser la position exacte de la ligne des 10 verges.

(b) La RA est aussi souvent utilisée au cinéma afin d'ajouter des acteurs virtuels à la scène réelle comme dans cette image tirée de *I, Robot* (2004).

FIG. 3.1 – Exemples d'utilisation de la RA à la télévision et au cinéma.

sont cependant généralement pas très précis (à moins que la pose de l'objet d'intérêt dans l'image de la caméra soit identique à l'une de celles retrouvées dans les images d'entraînement, ce qui est rarement le cas en pratique). De plus, cette approche globale n'est pas très efficace lorsque l'objet d'intérêt est partiellement occulté ou qu'il est dans une pose grandement différente de celles retrouvées dans les images d'entraînement.

D'autres méthodes d'estimation de la pose utilisent plutôt une approche plus locale

en ce qu'elles utilisent seulement des régions particulières de l'image de la caméra. Cette approche consiste en fait à rechercher dans un premier temps certaines caractéristiques (comme des coins par exemple) dans l'image de la caméra à l'aide d'outils spécialement conçus à cet effet (tel le détecteur de coins de Harris [50] pour poursuivre l'exemple). Ce sont ensuite ces points ou régions caractéristiques qui sont comparés au modèle connu de l'objet d'intérêt.

La plus grande difficulté de cette approche est de trouver des caractéristiques qui soient indépendantes des différentes variables entrant en jeu dans la composition de l'image (telle la pose de l'objet d'intérêt par exemple). Beaucoup de travail a été effectué à ce sujet. Shi et Tomasi [51] proposent par exemple un critère permettant de déterminer quels sont les meilleures caractéristiques à extraire d'un objet d'intérêt.

Les méthodes basées sur cette approche locale fournissent généralement des résultats plus précis que les méthodes basées sur l'approche globale quant à la pose 3D estimée des objets d'intérêt. De plus, puisqu'elles traitent les images de façon locale, les méthodes basées sur cette approche peuvent souvent fournir de bons résultats même si l'objet d'intérêt est partiellement occulté dans l'image de la caméra [37] [52].

Les méthodes basées sur l'approche locale sont pour ces raisons plus souvent utilisées pour estimer la pose 3D d'objets à partir d'images 2D.

Certaines méthodes [37] [53] utilisent des caractéristiques naturelles de l'objet d'intérêt. Cependant, différents facteurs (comme le bruit dans l'image par exemple) peuvent rendre la comparaison avec le modèle connu de l'objet d'intérêt très difficile.

Différentes approches ont été utilisées pour remédier à ce problème. State [41] propose par exemple de combiner cette méthode à un système de positionnement magnétique moins précis, mais plus robuste. Park, You et Neumann [35] [36] proposent pour leur part d'utiliser des marqueurs visuels artificiels afin d'initialiser et de faciliter par la suite la recherche de caractéristiques naturelles. Tout comme le système de positionnement magnétique proposé par State, ces marqueurs augmentent en plus la robustesse du système.

En effet, ces marqueurs sont construits de façon à contenir des caractéristiques facilement identifiables et associables au modèle connu de ces marqueurs. De tels marqueurs peuvent par exemple être composés de cercles de couleur faciles à repérer dans une image [39] [40]. Il est aussi possible d'exploiter le fait que les grands contrastes sont facilement repérables dans une image afin de construire des marqueurs [54] [55].

Le fait que les marqueurs puissent être construits sur mesure apporte plusieurs avantages. Premièrement, tel que nous venons de le voir, cela nous permet entre autres de les munir de caractéristiques très facilement repérables dans les images. De plus, cela nous permet d'avoir un modèle extrêmement précis de ces marqueurs, ce qui contribue à diminuer l'erreur commise dans l'estimation de leur pose 3D.

En fait, l'exploitation de marqueurs est tellement avantageuse que certaines méthodes d'estimation de la pose se limitent à cette façon de faire et n'utilisent pas du tout les caractéristiques naturelles[30].

3.3 Solution proposée

Les méthodes d'estimation de la pose 3D basées sur les caractéristiques naturelles ont fait d'énormes progrès dans les dernières années. Néanmoins, ces méthodes demandent encore généralement un temps d'exécution beaucoup plus long que celles basées sur l'utilisation de marqueurs artificiels. De plus, l'utilisation de caractéristiques naturelles est souvent une approche moins robuste que l'utilisation de marqueurs. Les résultats de l'estimation de la pose 3D des méthodes utilisant des marqueurs sont aussi généralement plus précis en raison de la précision du modèle de ces marqueurs et de la plus grande facilité à les comparer aux caractéristiques trouvées dans les images. Pour ces raisons, les applications nécessitant une estimation de pose 3D emploient souvent des méthodes utilisant des marqueurs. C'est aussi le choix que nous avons fait pour **Avatar**.

3.3.1 L'outil logiciel choisi

Il existe présentement une très grande quantité de méthodes d'estimation de la pose 3D à partir d'images 2D. De plus, des outils logiciels ont été développés pour bon nombre de celles-ci et ces outils sont parfois même librement accessibles aux programmeurs. Nous avons d'ailleurs décidé d'utiliser un tel outil logiciel pour implanter les parties traitant d'estimation de la pose 3D dans **Avatar**. L'outil en question est connu sous le nom de *ARToolKit* [30] et consiste en une librairie de fonctions en C pour le développement d'applications de RA. Les fonctions de *ARToolKit* permettent entre autres de calculer en temps réel la pose 3D de marqueurs relativement à la caméra les observant. *ARToolKit* a été choisi pour différentes raisons. En particulier, *ARToolKit* est basé sur des principes de vision par ordinateur relativement simples, ce qui lui confère une grande robustesse ainsi qu'une très grande rapidité d'exécution. De plus, il est pos-

sible, à partir des fonctions de *ARToolKit*, d'écrire un code pouvant être compilé sur n'importe quelle plate-forme sans aucune modification[2]. *ARToolKit* est aussi un outil logiciel à code ouvert. Est-il encore nécessaire de mentionner que ces deux dernières caractéristiques nous sont très chères et qu'elles ont grandement motivé notre choix ?

3.3.2 Utilisation de *ARToolKit* par Avatar

Tel que mentionné précédemment, la méthode d'estimation de la pose implémentée dans *ARToolKit* est basée sur l'utilisation de marqueurs. Cet outil logiciel ne permet donc pas d'estimer la pose 3D d'un objet quelconque dans une image. En fait, *ARToolKit* permet d'estimer uniquement la pose, relativement à la caméra les observant, de marqueurs spécifiques préalablement définis. Ces marqueurs sont d'ailleurs tous basés sur le même design. Il s'agit en fait de marqueurs plans carrés ne présentant aucune symétrie de rotation[3]. La figure 3.2 présente un exemple de marqueur pouvant être utilisé avec *ARToolKit*.

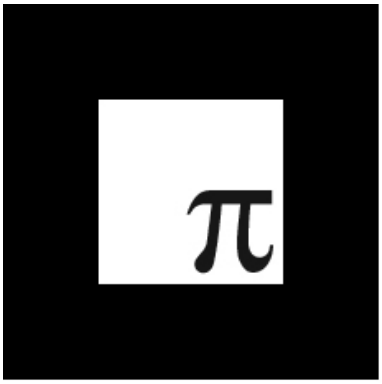


FIG. 3.2 – Exemple de marqueur pouvant être utilisé avec *ARToolKit*.

Le problème est que les acteurs du castelet réel dont on voudrait estimer la pose afin de contrôler les acteurs du castelet virtuel (visualisé avec **Avatar**) ne sont pas nécessairement de tels marqueurs. *ARToolKit* ne peut donc pas être utilisé directement pour évaluer la pose de ces acteurs. L'idée est alors de fixer des marqueurs sur les acteurs

[2]Cette possibilité de *ARToolKit* est cependant moins directe que dans le cas de VTK et *Qt*. L'écriture d'un code totalement portable sans aucune modification demande donc une certaine vigilance de la part du programmeur, mais est néanmoins possible.

[3]L'asymétrie de rotation des marqueurs est nécessaire afin d'éviter toute confusion. Celle-ci assure en effet qu'il n'existe qu'une seule solution possible à l'orientation du marqueur.

réels[4] et d'associer ces marqueurs[5] (plutôt que les acteurs réels) aux acteurs du monde virtuel. Notons que l'imposition de fixer des marqueurs sur les acteurs réels n'est pas une contrainte très importante. Ces marqueurs sont en effet très faciles à produire à l'aide d'une imprimante. De plus, on pourrait croire que la présence de tels marqueurs dans le castelet réel engendre une dégradation de l'aspect visuel de la scène. Or le castelet réel n'est utilisé que comme périphérique d'interaction avec les éléments constitutifs de la scène et le résultat de la mise en scène est plutôt visualisé dans le castelet virtuel dans lequel les marqueurs ne sont pas visibles. L'imposition des marqueurs n'est donc pas vraiment contraignante.

Ainsi, pour **Avatar**, ce sont ces marqueurs qui seront les acteurs réels et dont on estimera la pose. Ceux-ci étant fixés et donc toujours positionnés au même endroit sur les acteurs réels, la détermination de leur pose induit directement la pose de l'acteur sur lequel ils sont fixés. *ARToolKit* sera ainsi utilisé pour estimer la pose des marqueurs fixés sur les acteurs réels et cette pose sera ensuite conférée aux acteurs virtuels dans **Avatar**. De cette façon, toute scène réalisée à l'aide d'une maquette pourra être exactement reproduite dans **Avatar** et visualisée avec tous les avantages déjà mentionnés. La figure 3.3 illustre cette idée.

Afin d'être en mesure d'utiliser *ARToolKit* convenablement et d'intégrer ses fonctions d'estimation de la pose dans **Avatar**, il est important de d'abord comprendre son fonctionnement. Le lecteur non-familier avec le domaine de la vision par ordinateur est ici invité à consulter l'annexe A dans laquelle un certain formalisme mathématique nécessaire aux explications des concepts à venir est présenté.

3.4 Fonctionnement de *ARToolKit*

Plusieurs algorithmes sont impliqués dans le processus d'estimation de la pose de *ARToolKit*. L'explication en détail du fonctionnement précis de *ARToolKit* n'étant pas l'objectif de ce mémoire, seuls les algorithmes les plus importants seront couverts dans

[4]Le nombre de marqueurs à fixer sur un acteur réel dépend essentiellement de la liberté de mouvement de cet acteur relativement à l'emplacement de la caméra. Il suffit en fait de s'assurer qu'au moins un marqueur fixé sur cet acteur est toujours visible par la caméra. Lorsque la caméra est situé au dessus d'une scène, souvent un seul marqueur fixé sur le dessus de l'acteur est suffisant.

[5]Notons que tous les marqueurs utilisés dans une même scène doivent être différents les uns des autres afin d'éviter toute confusion.

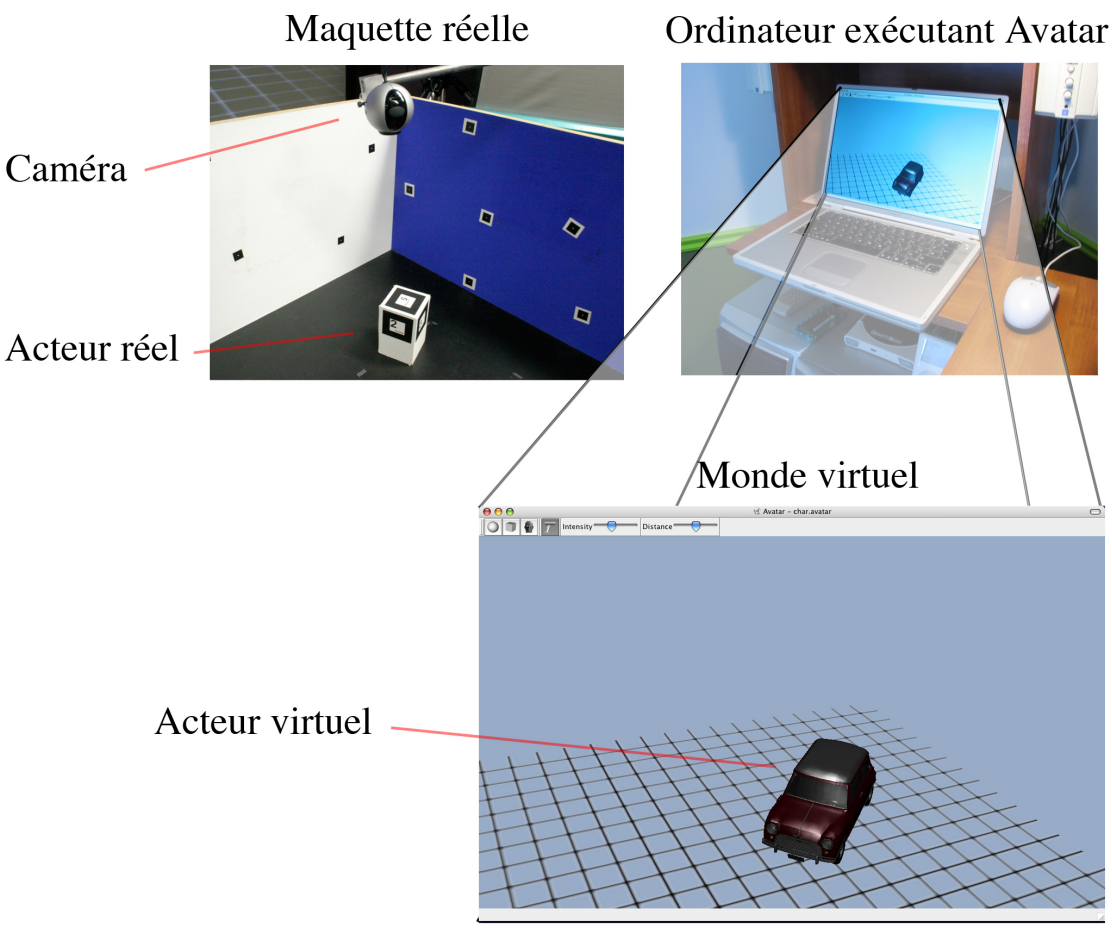


FIG. 3.3 – Une scène réelle peut être exactement reproduite de façon virtuelle dans **Avatar** par le biais de l'estimation de la pose des acteurs dans la maquette à partir d'images perçues par une caméra.

cette section[6].

3.4.1 Le repère de l'acteur

Tel que mentionné précédemment, *ARToolKit* ne peut estimer la pose que de certains marqueurs spécifiques et préalablement définis. En fait, afin d'être en mesure d'estimer la pose 3D d'un marqueur, *ARToolKit* doit avoir un modèle 3D bien précis de la géométrie de celui-ci dans un système de coordonnées particulier. Le système en question

[6]Il n'existe pas beaucoup de documentation décrivant le fonctionnement de *ARToolKit*. Ces explications ont principalement été déduites de la lecture du code de *ARToolKit* 2.61, lequel ne contient pratiquement aucun commentaire.

est appelé le *repère de l'acteur*. Ce système de coordonnées est en fait celui dans lequel est déjà définie la géométrie de l'acteur virtuel auquel est associé le marqueur. Il s'agit donc de définir la pose du marqueur dans le repère de cet acteur (voir figure 3.4).

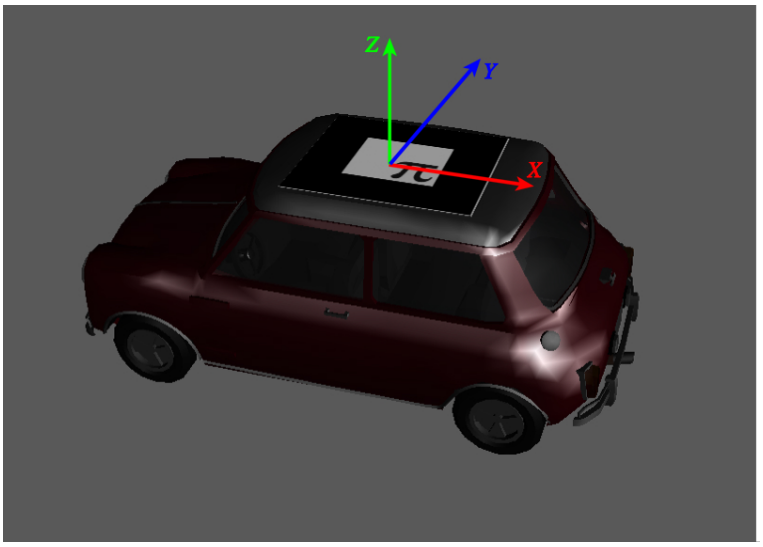


FIG. 3.4 – Exemple de modèle 3D d'un marqueur dans le repère de l'acteur auquel il est associé.

3.4.2 L'image d'entraînement normalisée

En plus de ce modèle géométrique, *ARToolKit* nécessite aussi un modèle de l'image du marqueur tel que perçu par la caméra. C'est en effet en comparant les images de la caméra avec ce modèle que *ARToolKit* peut identifier les marqueurs dans ces images. Cependant, l'image de ce marqueur dépend directement de sa pose relativement à la caméra. En effet, la transformation de projection de perspective[7] effectuée par la caméra ne préserve ni les angles ni les distances. Ainsi, même si le marqueur physique est carré, son image aura généralement la forme d'un quadrilatère quelconque.

Or, pour être en mesure de comparer l'image de la caméra avec un modèle de l'image du marqueur, nous aurons besoin d'une certaine cohérence. Pour cette raison, le modèle de l'image du marqueur tel que perçu par la caméra est en fait une image normalisée (carrée) de ce marqueur obtenue en transformant par homographie (voir la section A.3 en annexe) une image d'entraînement de ce marqueur[8]. Comme nous le verrons plus loin, ce modèle est utilisé afin de reconnaître le marqueur dans les images issues de la caméra.

[7] La projection de perspective est présentée à la section A.5 en annexe.
[8] Un programme fourni avec *ARToolKit* permet de créer de telles images normalisées.

3.4.3 La segmentation

Ceci étant dit, avant d'essayer de reconnaître un marqueur en le comparant avec les images normalisées (les modèles des images des marqueurs), il faut évidemment d'abord repérer les zones correspondant à des marqueurs dans les images de la caméra. La détection automatique d'objets dans une image peut parfois être très complexe. Cette problématique constitue même un champ d'étude en soi connu sous le nom de *segmentation* dans le domaine de la vision par ordinateur.

Il existe cependant des moyens de simplifier ce processus en considérant l'objectif particulier de la segmentation. Dans le cas de *ARToolKit* par exemple, cet objectif est d'estimer la pose des marqueurs. Comme nous le verrons plus loin, le calcul de cette pose n'est basé que sur les coordonnées image des pixels formant le contour du marqueur dans l'image et plus particulièrement sur celles des 4 coins de ce contour. L'objectif visé par la segmentation dans ce cas est donc la détection des points du contour.

Cette détection peut être facilitée par l'information que l'on détient *a priori* sur les marqueurs. En effet, une des caractéristiques des marqueurs utilisables avec *ARToolKit* (voir à cet effet la figure 3.2 en page 50) est qu'ils sont toujours imprimés en noir sur fond blanc. Cette caractéristique permet aux algorithmes de segmentation d'assurer une grande efficacité à faible coût de calcul. En effet, cette restriction a été élaborée en considérant le fait déjà mentionné que les forts contrastes sont facilement mis en évidence dans une image. C'est ce fait qui est exploité par les algorithmes de *ARToolKit*, lesquels peuvent, grâce à ce principe, détecter très rapidement les marqueurs dans l'image de la caméra par une technique appelée *segmentation par seuillage*.

La segmentation par seuillage

La segmentation par seuillage consiste à séparer tous les pixels d'une image en deux groupes distincts selon la valeur de leur intensité relativement à un certain seuil. Symboliquement, soient $p = (u, v)$ un pixel, $I : \mathbb{N}^2 \mapsto \mathbb{R}$ la fonction donnant l'intensité d'un pixel et s la valeur du seuil, alors la fonction de segmentation par seuillage est :

$$S : \mathbb{N}^2 \mapsto \mathbb{N}$$
$$S(p) = \begin{cases} 1 & \text{si } I(p) < s \\ 0 & \text{autrement} \end{cases}$$

Avec le seuil approprié, cette méthode permet de segmenter un objet d'intérêt dans

une image lorsqu'il existe un bon contraste entre les couleurs de cet objet et celles du reste de l'image.

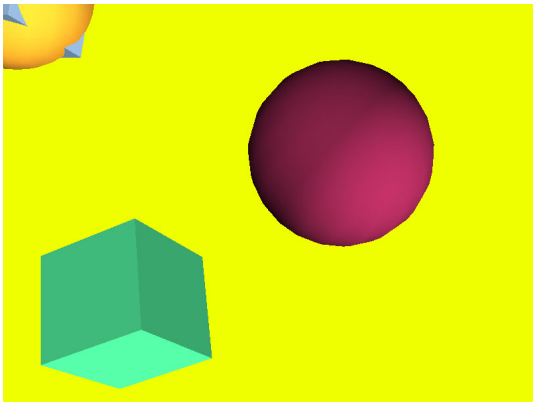
La figure 3.5(a) présente un exemple d'image dans laquelle l'objet d'intérêt, ici une sphère, contraste bien avec le reste de l'image. La couleur de la sphère en question est en effet beaucoup plus foncée que la couleur de l'arrière-plan et des autres objets de l'image. L'histogramme de l'intensité des pixels de cette image (Figure 3.5(c)) nous permet en fait de déterminer que les pixels correspondant à la sphère ont tous une intensité inférieure ou égale à 100 alors que les autres ont tous une intensité supérieure à cette valeur. Il est donc possible, dans ce cas particulier, de fixer le seuil à $s = 101$. La segmentation par seuillage produit alors une image binaire dans laquelle seuls les pixels correspondant à la sphère sont blancs (3.5(b)) et la segmentation est alors terminée.

Il n'est pas toujours évident de fixer la valeur du seuil à utiliser pour bien séparer les pixels, mais lorsque le contraste entre les couleurs de l'objet à segmenter et celles du reste de l'image est suffisamment important, une grande plage de valeurs est disponible. C'est pour maximiser cette plage que les marqueurs de *ARToolKit* sont imprimés en noir sur blanc. De plus, cette restriction fait en sorte qu'aucun histogramme n'est nécessaire pour déterminer la valeur du seuil à utiliser dans ce cas. En effet, ce dernier pourrait théoriquement être fixé à n'importe quelle valeur supérieure à 0 et inférieure à 255, soit les valeurs respectives des intensités du noir et du blanc dans une image. Afin de diminuer le risque d'erreur de classification engendré par l'imprécision des mesures effectuées par la caméra et afin de tenir compte du fait que l'éclairage de la scène n'est pas idéal et altère donc ces valeurs théoriques, la valeur du seuil ne devrait cependant pas être trop près d'une des bornes de cette plage. La valeur du seuil par défaut utilisée dans les algorithmes de *ARToolKit* est ainsi fixée à $s = 100$, ce qui donne généralement de bons résultats.

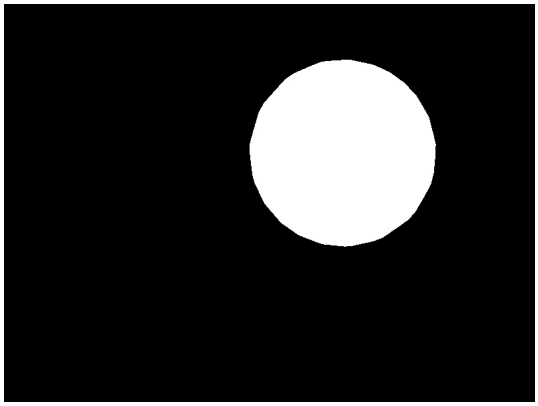
Cependant, notre expérience a démontré que l'éclairage de la scène nécessite parfois que ce seuil s soit ajusté. Cette possibilité allait donc devoir se retrouver dans **Avatar**.

Imprécisions et problématiques inhérentes à cette technique de segmentation

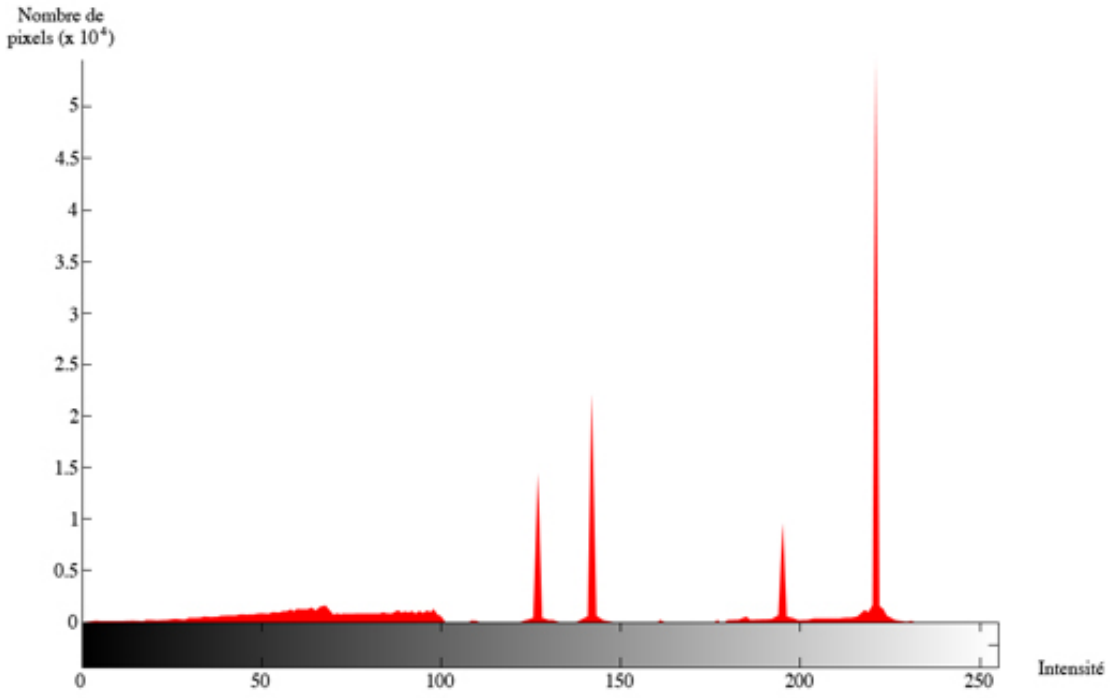
La technique de segmentation par seuillage présentée ici s'avère très efficace lorsqu'un bon contraste est présent entre les objets à segmenter et le reste de l'image. Cette condition n'est cependant pas respectée en général et il devient alors pratiquement impossible de trouver une valeur de seuil qui assure que tous, et seulement tous les pixels associés aux objets d'intérêt, soient sélectionnés. Il suffit en effet qu'une des couleurs des objets à segmenter ne contraste pas bien avec le reste de l'image pour que cette



(a) La sphère d'intérêt contraste bien avec le reste de l'image.

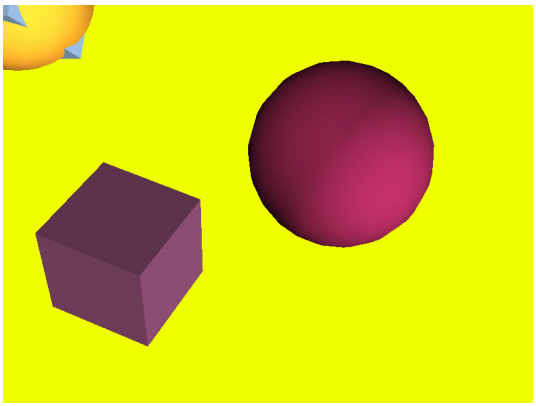


(b) Segmentation obtenue avec un seuil $s = 101$.

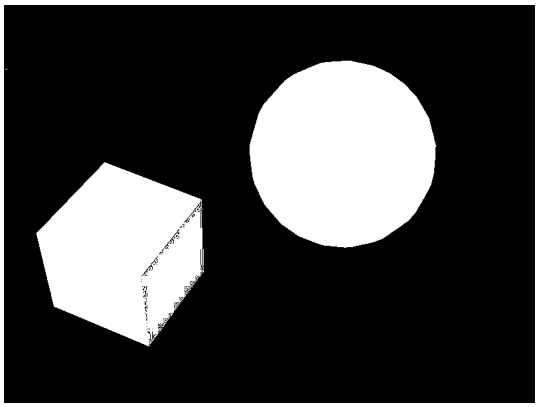


(c) Histogramme de l'intensité des pixels.

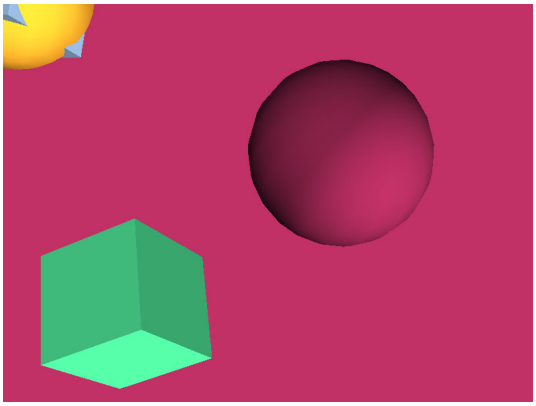
FIG. 3.5 – Exemples d'image se prêtant bien à la segmentation par seuillage.



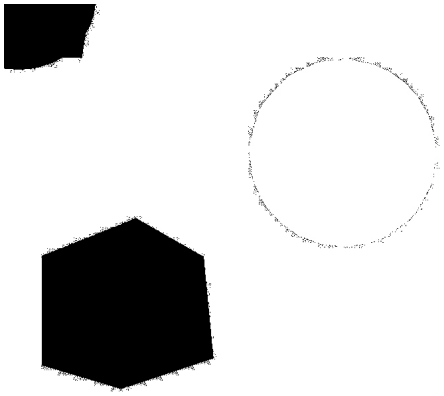
(a) Un cube ne contraste pas suffisamment avec la sphère d'intérêt.



(b) Segmentation obtenue avec un seuil $s = 101$.



(c) La couleur de l'arrière plan a une intensité trop près de celle de la sphère d'intérêt.



(d) Segmentation obtenue avec un seuil $s = 101$.

FIG. 3.6 – Exemples d'images ne se prêtant pas bien à la segmentation par seuillage. Dans ces images, la sphère d'intérêt a la même couleur que dans l'image 3.5. Ainsi, selon l'histogramme de la figure 3.5(c), le seuil s doit être supérieur à 100 afin de garantir que tous les pixels de la sphère d'intérêt fassent partie de la segmentation. Or, à $s = 101$ (la valeur minimale), des pixels n'appartenant pas à cette sphère sont déjà sélectionnés.

méthode de segmentation devienne très imprécise. Cette situation entraîne alors ce que l'on appelle des *fausses détections* (voir figure 3.6).

De telles imprécisions deviennent particulièrement problématiques lorsque la segmentation en question est utilisée dans un processus d'estimation de pose comme dans le cas de *ARToolKit*. L'obtention de bons résultats requiert en effet que les coordonnées image des points caractéristiques utilisés pour le calcul de la pose soient déterminées de façon très précise.

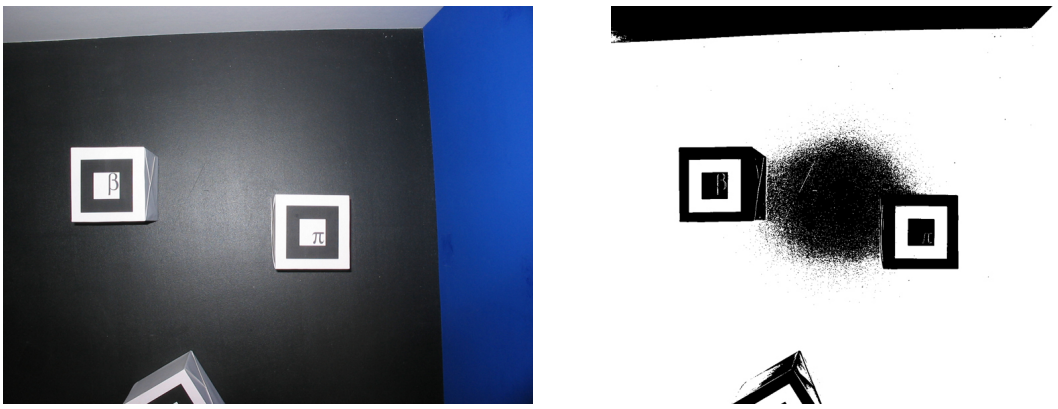
Dans le cas de *ARToolKit*, ce sont les points du contour extérieur du cadre carré des

marqueurs qui servent de points caractéristiques pour le processus d'estimation de la pose[9]. Il est donc important que ces points soient toujours précisément identifiés dans les images. À ce sujet, la couleur (noir) de ces points leur assure une intensité toujours inférieure au seuil s de segmentation. Ces points sont donc théoriquement[10] toujours inclus dans l'image binaire. Cependant, de fausses détections autour de ces points pourraient les rendre indiscernables parmi tous les points retenus dans la segmentation.

Pour remédier à ce problème, une bordure blanche est laissée autour des cadres noirs des marqueurs afin de créer un contraste qui assure que leur périmètre soit toujours bien délimité dans les images binaires. Cette procédure n'assure cependant aucunement que seuls les points caractéristiques désirés (les points formant le contour des marqueurs) se retrouveront dans les images binaires.

3.4.4 Raffinements

La figure 3.7 présente un exemple d'une image dans laquelle sont situés des marqueurs ainsi que l'image binaire obtenue de sa segmentation par seuillage. On remarque que plusieurs pixels autres que les points d'intérêts ont été retenus dans la segmentation.



(a) Image captée par une caméra présentant des marqueurs dont on voudrait estimer la pose.

(b) Segmentation obtenue avec un seuil $s = 128$.

FIG. 3.7 – Exemple d'image présentant des marqueurs dont on voudrait estimer la pose et segmentation par seuillage de cette image.

Les images binaires obtenues par seuillage doivent donc être analysées afin de raffiner

[9] La section 3.4.6 explique comment est obtenue la pose 3D des marqueurs à partir des coordonnées image des points du contour et plus particulièrement de leurs 4 coins.

[10] En pratique, certains types d'encre peuvent causer des réflexions spéculaires faisant en sorte que ces points ne sont pas retenus dans la segmentation par seuillage.

le résultat de la segmentation avant de procéder au calcul de la pose.

La première étape du processus de raffinement de la segmentation consiste à regrouper les pixels blancs de l'image binaire en groupes connexes. Ceci est réalisé par une procédure standard couramment utilisée en vision par ordinateur [34] connue sous le nom de *regroupement en composantes connexes*. Celle-ci consiste à parcourir l'image binaire et affecter d'une même étiquette tous les pixels blancs étant directement voisins[11].

Suite à cette procédure, l'image binaire est divisée en une foule de regroupements connexes. Chacun de ceux-ci correspond alors potentiellement à un marqueur. Or cela n'est pas le cas pour la majorité d'entre eux. Un certain traitement sur ces regroupements de pixels s'impose donc afin d'éliminer les fausses détections. Une analyse de l'aire couverte par chacun de ces regroupements permet dans un premier temps de discriminer une bonne partie des fausses détections.

Ensuite, le contour de chacun des regroupements restants est extrait. Les coordonnées image des points formant ces contours sont alors enregistrées et serviront ultérieurement.

Comme nous le verrons bientôt, l'estimation de la pose d'un marqueur est basée sur les coordonnées image de ses coins. La prochaine étape de raffinement de la segmentation consiste donc à déterminer quels sont les points du contour d'un regroupement qui correspondent à des coins. Lors de cette étape, plusieurs regroupements peuvent être éliminés de la segmentation. En effet, nous savons que les regroupements correspondant à des marqueurs sont des quadrilatères. Ainsi, tous les regroupements n'ayant pas exactement 4 coins peuvent être automatiquement éliminés. Il est à noter ici que les coordonnées image des coins des regroupements retenus (ceux ayant 4 coins) sont enregistrés pour référence ultérieure.

Après ces différentes étapes, plusieurs des regroupements initialement retenus par la segmentation par seuillage ont été éliminés. Cependant, nous pouvons améliorer encore davantage cette segmentation. En effet, nous savons que les regroupements correspondant à des marqueurs sont des quadrilatères. Nous avons d'ailleurs déjà utilisé cette information pour discriminer les regroupements n'ayant pas exactement 4 coins. Or le nombre de coins n'est pas la seule chose qui caractérise un quadrilatère.

[11] Il existe 2 types de *regroupement en composantes connexes*, nommés le *connectivité-4* et le *connectivité-8*. Ceux-ci se distinguent par leur définition du voisinage d'un pixel donné $p = (u, v)$. Dans le *connectivité-4*, seuls les 4 pixels formant un signe '+' avec ce pixel p (soient $p_1 = (u + 1, v)$, $p_2 = (u, v+1)$, $p_3 = (u-1, v)$ et $p_4 = (u, v-1)$) sont considérés ses voisins. Dans le cas du *connectivité-8*, ce sont plutôt les 8 pixels formant un carré autour de p qui sont considérés ses voisins. Les algorithmes de *ARToolKit* utilisent le *connectivité-8*.

En effet, un quadrilatère est aussi délimité par 4 côtés droits. Ainsi, la prochaine étape consiste à éliminer les regroupements ne satisfaisant pas ce critère. Pour cela, les algorithmes de *ARToolKit* tentent de faire correspondre 4 droites sur le contour de chacun des regroupements restants. Si l'erreur n'est pas trop importante, le regroupement est considéré comme un quadrilatère et les paramètres décrivant ces 4 droites sont enregistrés en mémoire (cette information sera utilisée plus tard dans le calcul de la pose du marqueur). De plus, les coordonnées image initiales des 4 coins de ce regroupement sont remplacées par les coordonnées (image) des points à l'intersection des 4 droites.

3.4.5 Transformation en image normalisée et comparaison avec les modèles

Les étapes précédentes ont permis de raffiner la segmentation initiale et de ne conserver que les regroupements de pixels ayant le plus de potentiel de correspondre à des marqueurs. Cependant, afin de s'assurer que chacun des regroupements conservés correspond bien à l'image d'un marqueur, ceux-ci doivent être comparés avec les modèles que nous avons préalablement définis (voir section 3.4.2).

Soit donc l'un de ces regroupements quadrilatères retenus. Les (nouvelles) coordonnées image des 4 coins de ce regroupement sont alors utilisées pour calculer l'homographie permettant de transformer ce quadrilatère en un carré de mêmes dimensions que les images d'entraînement normalisées. Cette image normalisée du quadrilatère est ensuite comparée par *template matching* avec les différentes images d'entraînement normalisées afin de déterminer si le quadrilatère en question est bel et bien l'image d'un marqueur. Cette opération nécessite de faire 4 comparaisons par image d'entraînement de marqueur, soit une par orientation[12]. Si l'image normalisée du quadrilatère est suffisamment semblable à l'image d'entraînement normalisée d'un des marqueurs, le quadrilatère est alors considéré comme l'image de ce marqueur. L'identité de ce marqueur ainsi que son orientation sont alors enregistrées et le processus d'estimation de la pose peut maintenant réellement commencer.

3.4.6 Estimation de la pose

Tel que mentionné à la section A.4 en annexe, tout objet du monde peut être exprimé dans le repère de la caméra et ce dernier peut en fait être considéré comme le repère

[12] N'oublions pas que les marqueurs ne présentent aucune symétrie de rotation.

du monde. Dans cette optique, la pose 3D du marqueur dans le monde est donc la pose 3D du marqueur dans le repère de la caméra. Or nous avons déjà la pose 3D de ce marqueur dans le repère de l'acteur (voir section 3.4.1). Ainsi, déterminer la pose 3D d'un marqueur (dans le repère du monde) revient à trouver la relation qui existe entre le repère de l'acteur dans lequel ce marqueur particulier est décrit et le repère de la caméra.

Cette relation s'exprime par une transformation mathématique appelée *changement de base*. Un changement de base est généralement réalisé par une transformation rigide, c'est-à-dire par la composition d'une rotation $\mathbf{R}$ et d'une translation $\mathbf{T}$.

Ainsi, soit ${}^{A}P = [{}^{A}x, {}^{A}y, {}^{A}z]^T$ un point exprimé dans le repère A de l'acteur. Les coordonnées ${}^{C}P = [{}^{C}x, {}^{C}y, {}^{C}z]^T$ de ce même point dans le repère C de la caméra (avant la projection de perspective évidemment) sont données par :

$$ {}^{C}P = \mathbf{R}\ {}^{A}P + \mathbf{T} \tag{3.1} $$

où

$$ \mathbf{R} = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{pmatrix} = \begin{pmatrix} \mathbf{r_1} & \mathbf{r_2} & \mathbf{r_3} \end{pmatrix} $$

est la matrice de rotation décrivant l'orientation du repère de l'acteur dans le repère de la caméra et où $\mathbf{T} = [t_1, t_2, t_3]^T$ est l'origine du repère de l'acteur exprimé dans le repère de la caméra. À noter que les vecteurs $\mathbf{r_1}$, $\mathbf{r_2}$ et $\mathbf{r_3}$ composant les colonnes de $\mathbf{R}$ sont en fait les vecteurs de base du repère de l'acteur exprimé dans le repère de la caméra.

La figure 3.8 illustre cette relation de changement de base.

Il s'agit donc maintenant de déterminer quels sont les paramètres de rotation et de translation constituant $\mathbf{R}$ et $\mathbf{T}$. Commençons d'abord par la rotation.

Estimation de la rotation

C'est ici qu'entrent en jeu les paramètres précédemment enregistrés des droites correspondant au contour des marqueurs (voir p. 60).

Soient

$$ \begin{cases} a_1 u + b_1 v + c_1 &= 0 \\ a_2 u + b_2 v + c_2 &= 0 \end{cases} \tag{3.2} $$

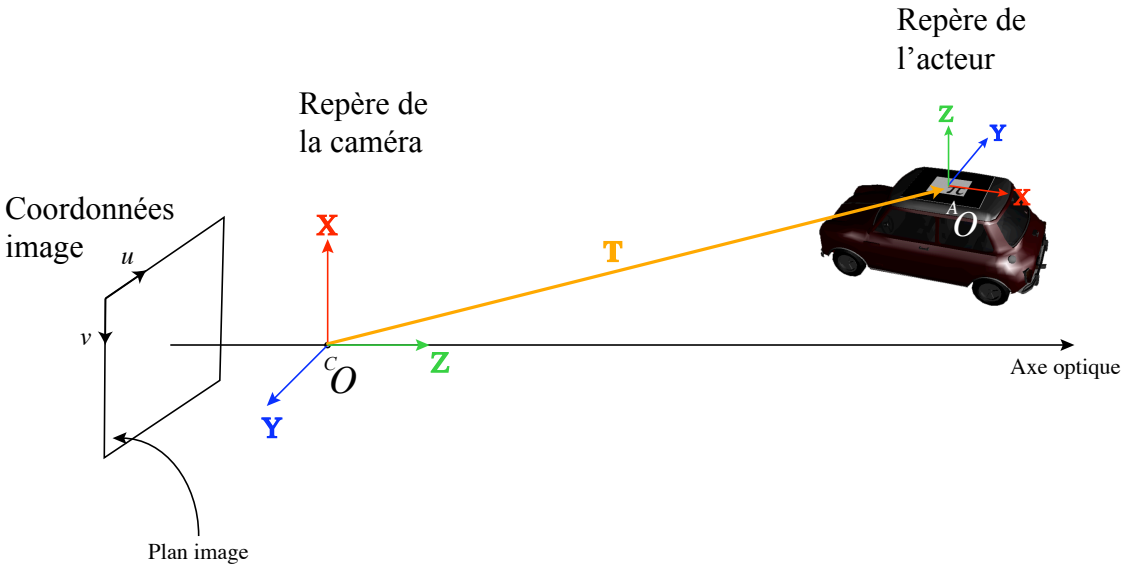


FIG. 3.8 – Relation entre le repère de l'acteur et le repère de la caméra (changement de base).

les équations (en coordonnées image) des droites correspondant à 2 côtés parallèles d'un marqueur. Selon la transformation de projection de perspective[13], nous avons

$$\begin{cases} u &= \alpha \frac{{}^{c}x}{{}^{c}z} - \alpha \cot\theta \frac{{}^{c}y}{{}^{c}z} + u_0 \\ v &= \frac{\beta}{\sin\theta} \frac{{}^{c}y}{{}^{c}z} + v_0. \end{cases} \tag{3.3}$$

Ainsi, les équations (3.2) (exprimées en coordonnées image) peuvent être réécrites de la façon suivante dans le repère de la caméra :

$$\begin{cases} a_1 \left(\alpha \frac{{}^{c}x}{{}^{c}z} - \alpha \cot\theta \frac{{}^{c}y}{{}^{c}z} + u_0 \right) + b_1 \left(\frac{\beta}{\sin\theta} \frac{{}^{c}y}{{}^{c}z} + v_0 \right) + c_1 &= 0 \\ a_2 \left(\alpha \frac{{}^{c}x}{{}^{c}z} - \alpha \cot\theta \frac{{}^{c}y}{{}^{c}z} + u_0 \right) + b_2 \left(\frac{\beta}{\sin\theta} \frac{{}^{c}y}{{}^{c}z} + v_0 \right) + c_2 &= 0. \end{cases} \tag{3.4}$$

En réécrivant ces équations en fonction de ${}^{c}x$, ${}^{c}y$ et ${}^{c}z$, nous obtenons les équations de deux plans dans le repère de la caméra :

$$\begin{cases} a_1\alpha {}^{c}x + \left(-a_1\alpha \cot\theta + \frac{b_1\beta}{\sin\theta} \right) {}^{c}y + (a_1u_0 + b_1v_0 + c_1) {}^{c}z &= 0 \\ a_2\alpha {}^{c}x + \left(-a_2\alpha \cot\theta + \frac{b_2\beta}{\sin\theta} \right) {}^{c}y + (a_2u_0 + b_2v_0 + c_2) {}^{c}z &= 0 \end{cases} \tag{3.5}$$

[13] Voir à ce sujet l'équation (A.4) en annexe.

ou, plus simplement,

$$\begin{cases} A_1{}^C x + B_1{}^C y + C_1{}^C z &= 0 \\ A_2{}^C x + B_2{}^C y + C_2{}^C z &= 0. \end{cases} \tag{3.6}$$

Ces équations décrivent en fait les deux plans du repère de la caméra dont la projection sur le plan image par la transformation de projection de perspective (3.3) résulte en les deux droites parallèles décrites par l'équation (3.2) et qui correspondent à 2 côtés parallèles d'un marqueur.

Les vecteurs normaux à ces 2 plans sont donc respectivement (toujours dans le repère de la caméra)

$$\begin{cases} n_1 &= (A_1, B_1, C_1) \\ n_2 &= (A_2, B_2, C_2). \end{cases} \tag{3.7}$$

La figure 3.9 illustre cette situation.

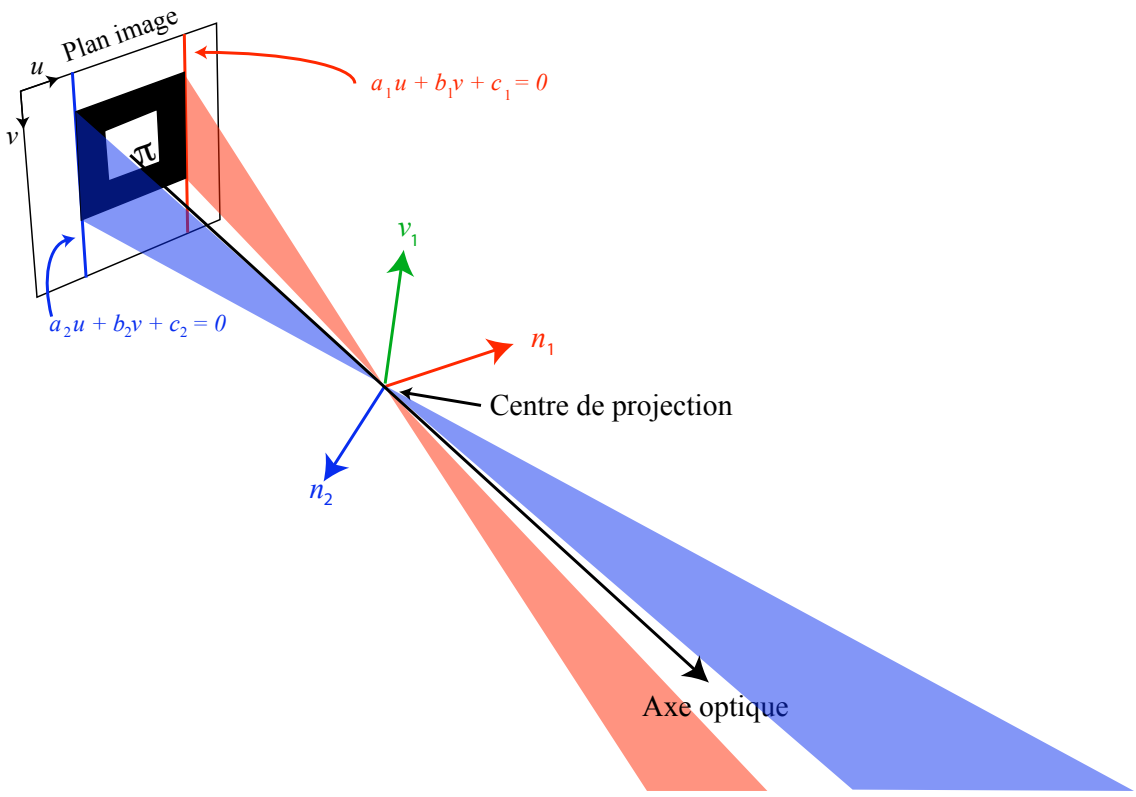


FIG. 3.9 – Illustration de deux plans de normale n_1 et n_2 respectivement (dans le repère de la caméra) dont la projection sur le plan image par la transformation de projection de perspective (3.3) résulte en deux droites parallèles qui correspondent à 2 côtés parallèles d'un marqueur.

Ainsi, le vecteur

$$v_1 = \frac{n_1 \times n_2}{\left\| n_1 \times n_2 \right\|}$$

est nécessairement un vecteur unitaire orthogonal aux vecteurs n_1 et n_2, c'est donc dire un vecteur unitaire dans la même direction (dans le repère de la caméra) que les 2 cotés parallèles considérés.

Comme nous ne nous intéressons présentement uniquement qu'à la composante *rotation* du changement de base, nous pouvons supposer pour l'instant que le repère de l'acteur n'est séparé du repère de la caméra que par une rotation seulement. De cette façon, les origines des 2 repères coïncident et le vecteur v_1 peut alors être considéré comme un vecteur de base du repère de l'acteur exprimé dans le repère de la caméra.

En répétant cette procédure pour l'autre paire de côtés parallèles du marqueur, nous obtenons de la même façon un vecteur unitaire v_2 dans la même direction (toujours dans le repère de la caméra) que ces autres côtés parallèles du marqueur. Tout comme v_1, ce vecteur v_2 peut lui aussi être considéré comme un vecteur de base du repère de l'acteur exprimé dans le repère de la caméra.

Maintenant, puisque les directions des 2 paires de côtés parallèles du marqueur sont orthogonales entre elles, le vecteur v_2 est théoriquement orthogonal au vecteur v_1. Cependant, le traitement d'image et les calculs effectués jusqu'à présent ont généralement introduit certaines erreurs. Pour remédier à cette situation et s'assurer que le repère de l'acteur est bel et bien un système orthogonal, définissons r_1 et r_2, deux vecteurs unitaires orthogonaux dans le plan contenant v_1 et v_2 et définis tel qu'illustré à la figure 3.10.

Toujours en supposant que le repère de l'acteur n'est séparé du repère de la caméra que par une rotation seulement, les vecteurs r_1 et r_2 peuvent être considérés comme 2 vecteurs de base du repère de l'acteur exprimés dans le repère de la caméra. Ainsi, en posant $r_3 = r_1 \times r_2$, nous obtenons la matrice de rotation recherchée :

$$\mathbf{R} = \begin{pmatrix} \mathbf{r_1} & \mathbf{r_2} & \mathbf{r_3} \end{pmatrix}.$$

Estimation de la translation

Maintenant que nous connaissons la matrice décrivant la rotation du changement de base entre le repère de l'acteur et celui de la caméra, nous sommes en mesure d'évaluer

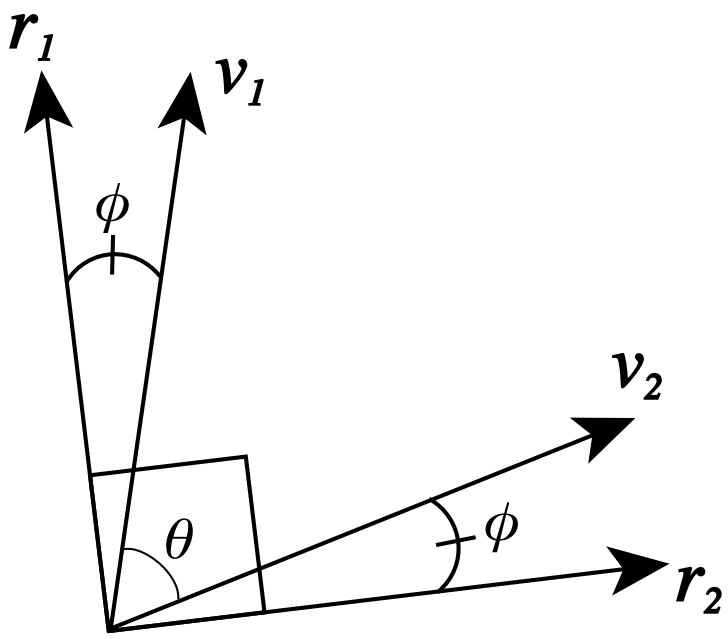


FIG. 3.10 – Définition des vecteurs orthogonaux r_1 et r_2 dans le plan contenant v_1 et v_2. Soit θ l'angle entre v_2 et v_1 et soit $\phi = \frac{1}{2}\left(\frac{\pi}{2} - \theta\right)$, r_1 est défini en appliquant une rotation de ϕ à v_1 et r_2 est défini en appliquant une rotation de $-\phi$ à v_2.

la composante *translation* de ce changement de base. En effet, soient $[{}^{A}x_i, {}^{A}y_i, {}^{A}z_i]^T$ $(i = 1, 2, 3, 4)$ les coordonnées des 4 coins du marqueur dans le repère de l'acteur. L'équation du changement de base (3.1) nous permet alors d'exprimer ces points dans le repère de la caméra :

$$\begin{bmatrix} {}^{C}x_i \\ {}^{C}y_i \\ {}^{C}z_i \end{bmatrix} = \mathbf{R} \begin{bmatrix} {}^{A}x_i \\ {}^{A}y_i \\ {}^{A}z_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{3.8}$$

où t_1, t_2 et t_3 sont inconnus.

De plus, la matrice $\mathbf{M}$ de projection de perspective[14] (obtenue par le calibrage de la caméra) nous permet de déterminer les coordonnées image de la projection de ces 4

[14]Nous parlons ici de la matrice $\mathbf{M}$ de l'équation (A.5) en annexe.

points du repère de la caméra :

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \frac{1}{w_i} M \begin{bmatrix} {}^{C}x_i \\ {}^{C}y_i \\ {}^{C}z_i \end{bmatrix} \tag{3.9}$$

Nous avons donc, en combinant les équation (3.8) et (3.9),

$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \frac{1}{w_i} M \left(\mathbf{R} \begin{bmatrix} {}^{A}x_i \\ {}^{A}y_i \\ {}^{A}z_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \right) \tag{3.10}$$

Or nous connaissons déjà $[{}^{A}x_i, {}^{A}y_i, {}^{A}z_i]^T$, les coordonnées des 4 coins du marqueur dans le repère de l'acteur. De plus, nous connaissons aussi u_i et v_i, les coordonnées image de la projection de ces points. Les seules inconnues de l'équation (3.10) sont donc t_1, t_2 et t_3. En combinant les 2 équations (3.10) fournies par chacun des 4 coins, nous obtenons un système de 8 équations à trois inconnues. Ce système peut alors être résolu par la méthode des moindres carrés [34].

La résolution de ce système nous donne alors la composante *translation* du changement de bases recherché, ce qui complète ainsi son calcul.

Afin de pallier aux erreurs introduites par le traitement d'images et les différents calculs numériques effectués, les algorithmes de *ARToolKit* procèdent ensuite à l'optimisation de la transformation obtenue. Cette optimisation est réalisée par un processus itératif consistant essentiellement à minimiser la somme des erreurs entre les coordonnées image prédites par l'équation (3.10) et les coordonnées image réelles de la projection des 4 coins du marqueur.

La transformation résultant de cette optimisation est alors précisément la transformation désirée par **Avatar**. En effet, comme nous l'avons vu précédemment, la pose d'un marqueur est définie dans le repère de l'acteur auquel il est associé. Or la pose de l'acteur virtuel est également définie dans ce repère. Ainsi, la transformation obtenue nous permet de calculer la pose de l'acteur virtuel dans le repère de la caméra, c'est-à-dire dans le repère du monde.

La figure 3.11 résume les divers résultats obtenus dans cette section et met en évidence leur relation avec les équations permettant de les obtenir.

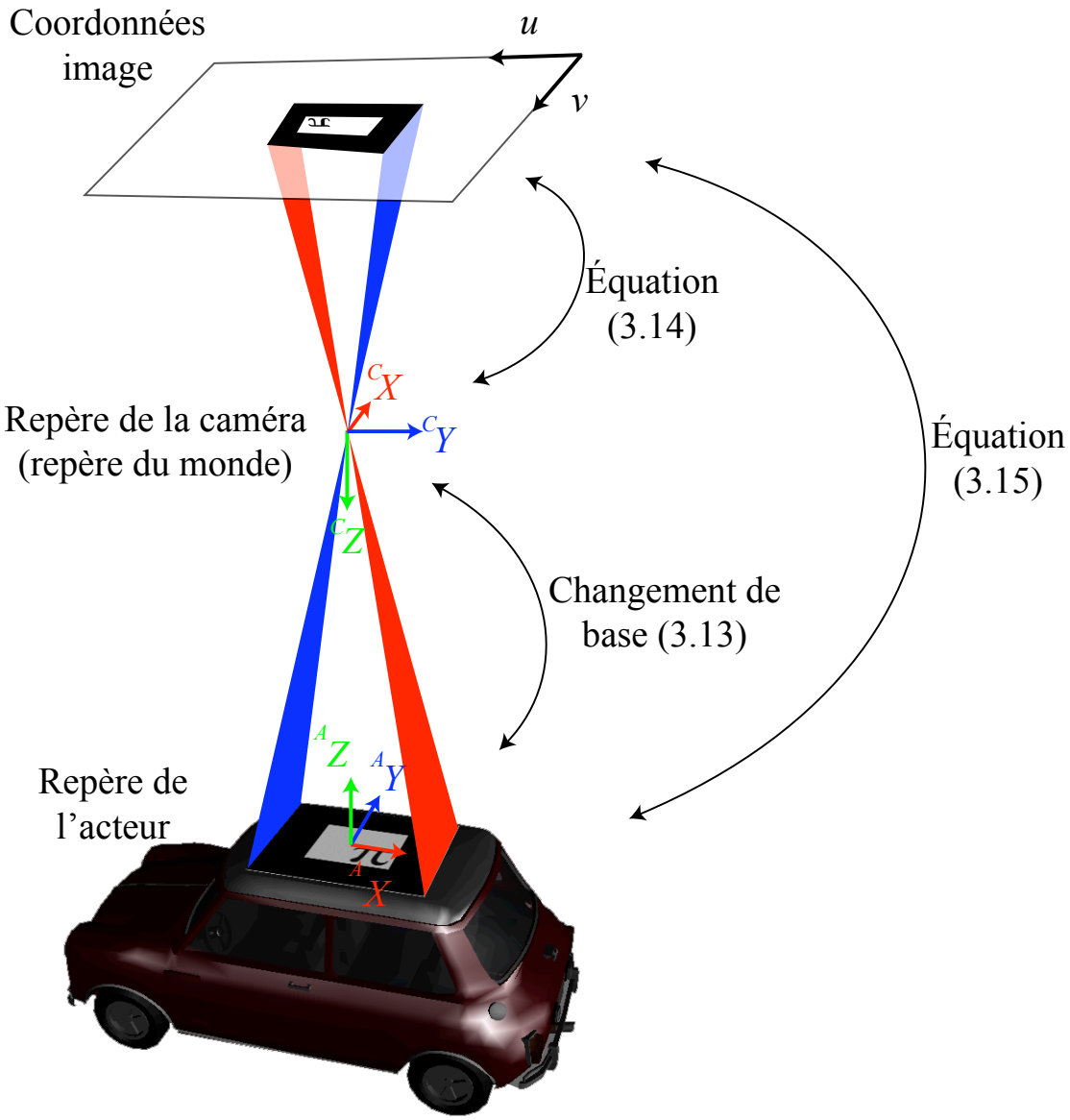


FIG. 3.11 – La relation entre le repère de la caméra (le repère du monde) et les coordonnées image est donnée par l'équation (3.9). Cette équation est obtenue du calibrage de la caméra. De plus, comme nous l'avons vu, nous pouvons estimer les inconnues de l'équation (3.10) donnant la relation entre le repère de l'acteur et les coordonnées image. Nous pouvons alors en déduire l'équation (3.8) du changement de base entre le repère de l'acteur et celui de la caméra. Ce changement de base nous permet finalement de connaître la pose de l'acteur (définie dans le repère de l'acteur) dans le repère de la caméra, c'est-à-dire le repère du monde.

3.5 Présentation d'Avatar, l'application de réalité virtuelle collée à la réalité

Comme nous venons de le voir, la transformation rigide calculée par *ARToolKit* permet d'affecter les acteurs dans la scène virtuelle d'une pose identique à celles des marqueurs dans la scène réelle. Cela nous permet donc de recréer, dans un monde virtuel, toute scène créée dans le monde réel. Les metteurs en scène peuvent ainsi élaborer leur scène avec une maquette tel qu'ils ont l'habitude de le faire et visualiser en temps réel un résultat beaucoup plus révélateur dans un monde virtuel.

Afin d'illustrer ces propos, un autre exemple d'utilisation d'**Avatar** exploitant ces nouvelles possibilités est maintenant présenté. Dans cet exemple, nous élaborerons une scène de poursuite de voitures.

Tel que mentionné précédemment, le fait que la scène soit visualisée dans un monde virtuel permet aux metteurs en scène d'utiliser une maquette générique épurée et réutilisable pour faire leur travail. En effet, la maquette ne sert maintenant que de périphérique d'interaction et non d'outil de visualisation. Cela convient d'ailleurs parfaitement au projet de Castelet Électronique puisque dans ce cas, la maquette (le castelet réel muni d'un plancher robotisé et d'un éclairage à base de fibres optiques) a été développée pour la mise en scène de plus d'une œuvre.

Dans la même lignée, rien n'oblige les acteurs réels à ressembler aux acteurs qu'ils représentent dans la scène virtuelle. En effet, tout ce qui importe est qu'un ou plusieurs marqueurs soient attachés à ces acteurs réels. En fait, le marqueur lui-même est suffisant comme acteur réel et l'objet sur lequel ce marqueur est fixé ne sert que de support physique facilitant sa manipulation.

La figure 3.12 présente une photo de la maquette de la mise en scène de poursuite de voitures. Les prismes présents dans la maquette sont en fait les acteurs réels représentant les 2 voitures.

Une fois que nous avons déterminé quels acteurs réels seront utilisés et que des marqueurs ont été fixés sur chacun de ceux-ci, la prochaine étape dans notre processus de mise en scène consiste à créer des fichiers décrivant chacun de ces marqueurs. Plus concrètement, pour chacun des marqueurs, l'utilisateur doit créer un fichier dans lequel est inscrit un chemin vers l'image d'entraînement normalisée de ce marqueur (créée à partir d'un programme fourni avec *ARToolKit*) ainsi que des informations décrivant sa pose dans le repère de l'acteur.

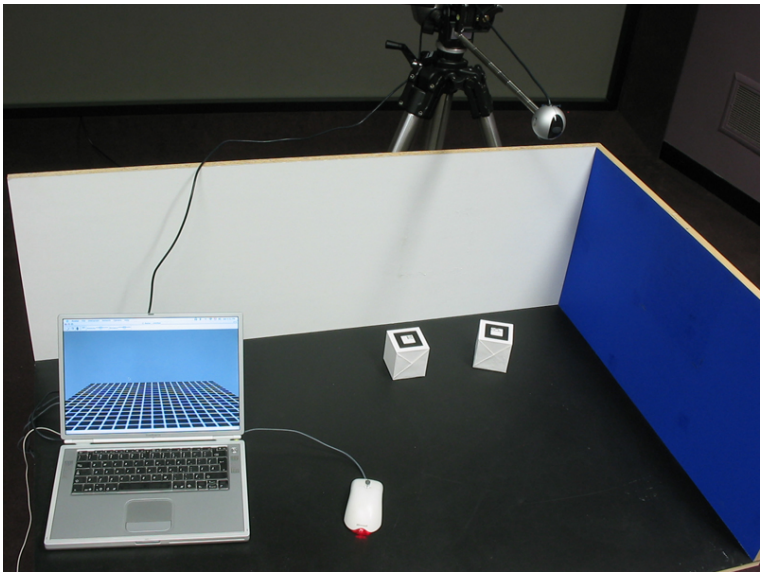


FIG. 3.12 – Exemple de maquette générique pouvant être utilisée avec **Avatar**.

Les prochaines étapes du processus de mise en scène à partir d'**Avatar** sont ensuite exactement les mêmes que celles présentées à la section 2.3. Cependant, lors de l'ajout d'un acteur, **Avatar** demande maintenant à l'utilisateur si ce nouvel acteur est associé à un marqueur dans le monde réel. Si tel est le cas, **Avatar** demande alors à l'utilisateur de lui indiquer quel fichier décrit le marqueur en question. Suite à cela, l'acteur est ajouté à l'origine du monde virtuel, comme dans le cas où il n'est pas associé à un marqueur. L'utilisateur peut alors interagir avec cet acteur exactement comme il pouvait le faire dans la version précédente de l'application. Le processus de création de monde virtuel est donc pratiquement identique à celui présenté à la section 2.3. Afin d'éviter une redondance inutile, ce processus ne sera pas répété ici.

La figure 3.13 présente ainsi la scène de poursuite de voitures une fois que les 2 acteurs ont été ajoutés au monde virtuel. Ces acteurs ont chacun été associés à un marqueur particulier lors de leur ajout au monde virtuel.

Le lecteur remarquera sans doute un nouvel élément dans l'IUG d'**Avatar**, à savoir une barre d'outils constituant en fait un raccourci vers certains éléments des menus. Cette barre contient des icônes associées aux actions les plus couramment posées lors d'une utilisation typique d'**Avatar**. L'une de ces icônes est d'ailleurs associée à la nouvelle fonction d'**Avatar** permettant d'utiliser une caméra afin d'estimer la pose des acteurs de la scène réelle et de positionner les acteurs virtuels en concordance[15].

[15] Cette fonctionnalité d'**Avatar** est aussi accessible par le menu *Interaction* ainsi que par un raccourci clavier.

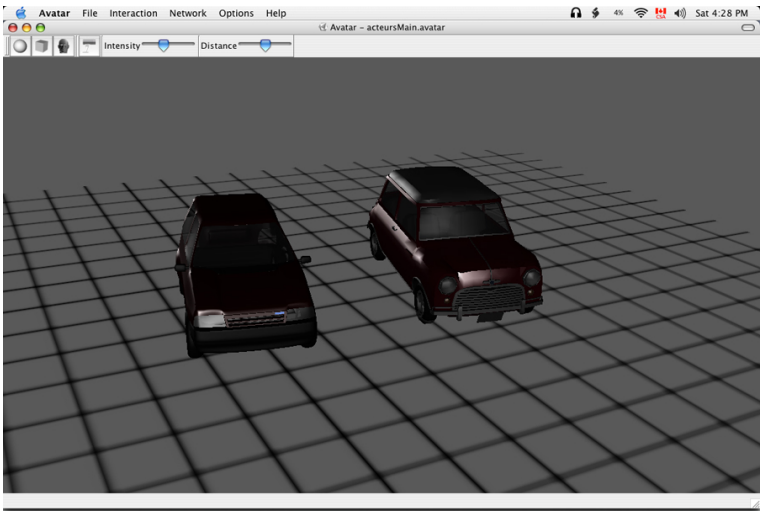


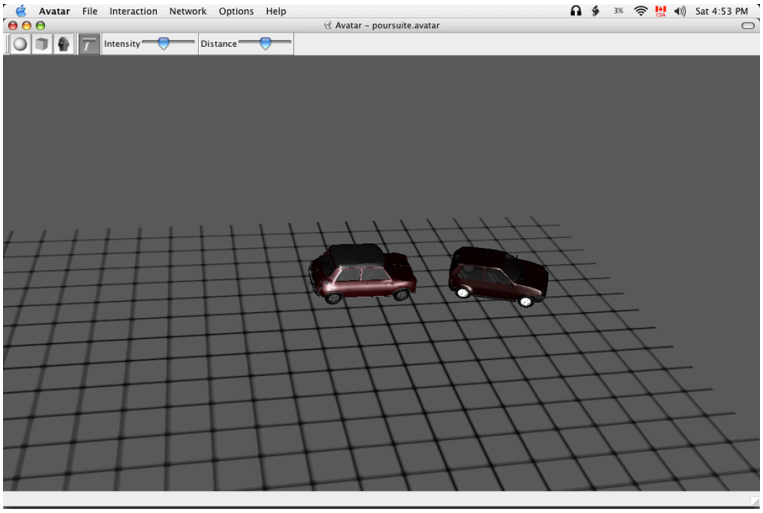
FIG. 3.13 – Scène de poursuite de voitures en élaboration avec **Avatar**.

Lorsque l'utilisateur choisit d'utiliser cette fonctionnalité, **Avatar** tente d'accéder à la caméra branchée à l'ordinateur. Si aucune caméra n'est détectée par **Avatar**, un message d'erreur est alors affiché à l'écran afin de prévenir l'utilisateur du problème rencontré. Dans le cas contraire, **Avatar** utilise *ARToolKit* afin d'estimer la pose des marqueurs présents dans la scène réelle tel qu'expliqué à la section précédente et positionne ensuite les acteurs virtuels correspondant à ces marqueurs de la même façon dans le monde virtuel.

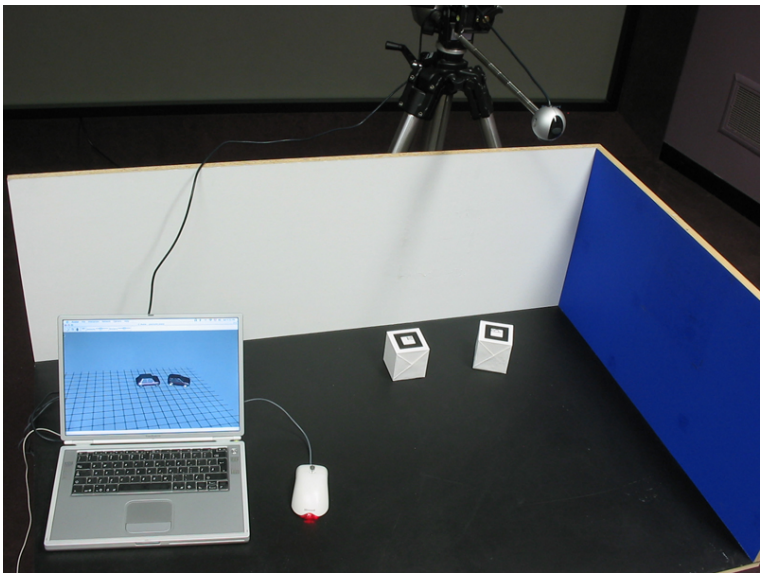
À titre d'illustration, la figure 3.14(a) présente le monde virtuel correspondant à notre scène réelle (figure 3.14(b)).

Lors de l'utilisation de cette nouvelle fonctionnalité, la maquette sert de périphérique afin de contrôler la pose des acteurs du monde virtuel associés à des marqueurs. Il est cependant toujours possible d'ajouter un acteur sans l'associer à un marqueur. Ainsi, nous pourrions ajouter un décor à la scène virtuelle et aller le positionner à l'aide de la souris tel que nous l'avons fait dans notre exemple précédent. Cependant, le décor de la scène peut aussi être considéré comme un acteur, au même titre que tous les autres acteurs. Ainsi, nous pouvons fixer simplement un marqueur sur le plancher de la maquette par exemple et associer ce marqueur à l'acteur *décor*. De cette façon, celui-ci sera automatiquement positionné au bon endroit dans le monde virtuel (voir figure 3.15).

Dans la figure 3.15(a), la caméra virtuelle (nous permettant de visualiser le monde virtuel) a été positionnée dans environ la même pose que l'appareil photo qui a photographié la scène de la figure 3.15(b). Il est bon de noter que même si la pose des



(a) Monde virtuel dans lequel les acteurs virtuels (les voitures) se virent attribuer la même pose que les acteurs réels.

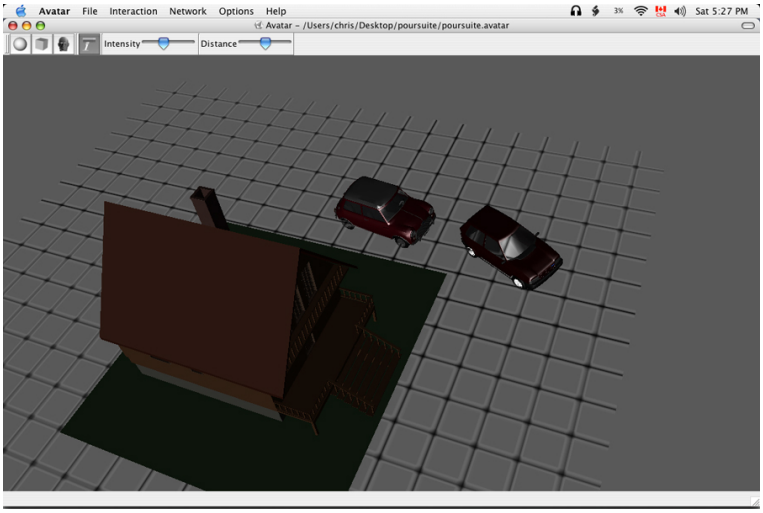


(b) Scène réelle utilisée pour contrôler le monde virtuel.

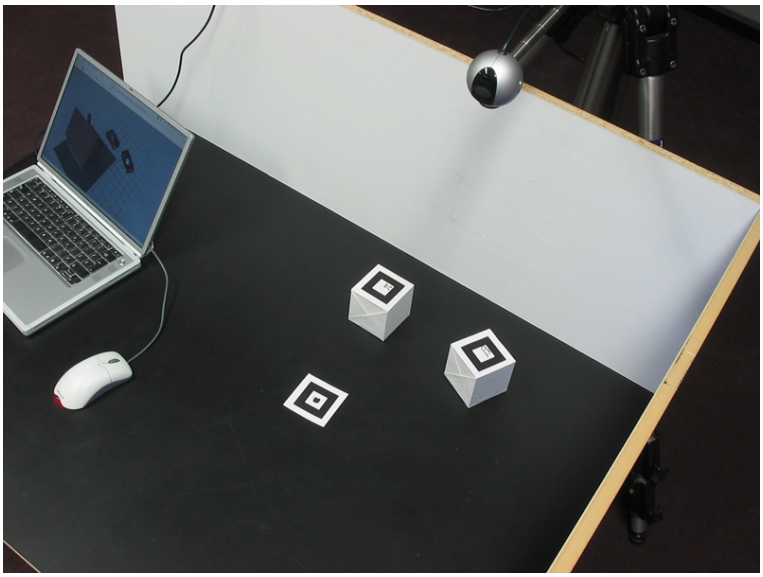
FIG. 3.14 – Exemples d'utilisation d'une caméra afin de contrôler la pose des acteurs virtuels dans **Avatar**.

acteurs est déterminée par *ARToolKit*, la caméra observant ces acteurs peut elle être positionnée n'importe où dans le monde virtuel.

L'expérience prouve qu'il n'est pas toujours évident de se servir de cette nouvelle fonctionnalité d'**Avatar** et que plusieurs facteurs peuvent influencer son fonctionnement. De plus, il est très difficile de trouver la cause d'un problème lié à cette fonc-



(a) Monde virtuel dont le décor a été positionné par l'estimation de la pose d'un marqueur.



(b) Scène réelle utilisée pour contrôler le monde virtuel. Dans cette scène, un des marqueurs est associé au décor.

FIG. 3.15 – Exemples d'utilisation d'un marqueur afin de positionner les objets du décor dans **Avatar**.

tionnalité lorsque l'on ne connaît pas l'image fournie à *ARToolKit*. Ainsi, afin d'aider l'utilisateur à régler les problèmes qu'il pourrait rencontrer, un nouveau point de vue pré-enregistré a été ajouté dans le menu d'**Avatar**. Celui-ci correspond en fait au point de vue de la caméra réelle. Lorsque ce point de vue est sélectionné, la caméra virtuelle se voit attribuer la même pose que la caméra réelle et l'image de cette dernière est ajoutée en arrière-plan, comme dans une application de RA (voir figure 3.16).

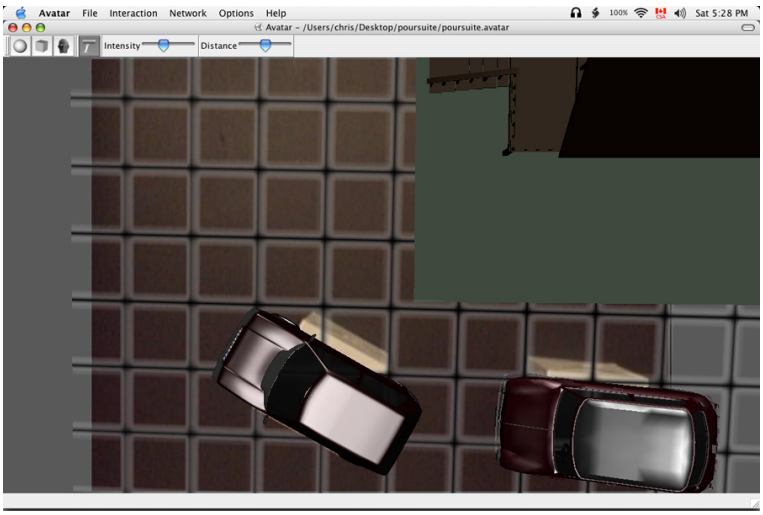


FIG. 3.16 – **Avatar** permet de visualiser en arrière-plan l'image fournie à *ARToolKit*, à la manière d'une application de RA.

Cela permet à l'utilisateur de s'assurer que l'image fournie à l'application est bel et bien conforme à l'utilisation que celle-ci en fait. Cela permet entre autres de vérifier que les marqueurs sont bien visibles par la caméra, que l'éclairage ne cause pas de réflexions spéculaires sur les marqueurs (ce qui empêche leur détection), que les contrastes sont assez importants ou encore que l'intensité moyenne de l'image n'est pas trop faible. À ce sujet, la barre d'outil d'**Avatar** contient un ascenseur permettant d'ajuster le seuil de segmentation (voir section 3.4.3).

Il est bon de réaliser que pour que les objets virtuels paraissent faire partie intégrante du monde réel dans l'image de la figure 3.16, les paramètres intrinsèques de la caméra virtuelle ont dû être modifiés. En effet, par défaut, la caméra virtuelle est une caméra idéale, ce qui n'est évidemment pas le cas de la caméra réelle. Ainsi, lorsque ce modèle de caméra idéale est utilisé pour visualiser les objets virtuels sur un arrière-plan constitué d'une image de la caméra réelle (non idéale), le mariage réel-virtuel n'est pas satisfaisant. Pour remédier à cette situation, les paramètres intrinsèques de la caméra réelle (estimés à l'aide du programme de calibrage prémentionné fourni avec *ARToolKit*) doivent être attribués à la caméra virtuelle.

3.6 Synthèse récapitulative

Comme nous l'avons vu au chapitre précédent, **Avatar** est d'abord et avant tout une application de RV et à ce titre, la première version d'**Avatar** avait atteint tous les

objectifs fixés au départ. Cependant, cette application de RV qui peut très bien être utilisée à d'autres fins a tout de même à la base été élaborée afin de rencontrer les objectifs spécifiques et particuliers du projet de Castelet Électronique. Or, un des principaux objectifs de ce projet était de faire en sorte que les metteurs en scène puissent utiliser le castelet réel du LANTISS afin de contrôler le castelet virtuel. Comme nous l'avons vu, cet objectif a maintenant été atteint avec cette nouvelle version de l'application. En effet, **Avatar** offre maintenant à l'usager la possibilité d'utiliser des acteurs réels afin de contrôler les acteurs du monde virtuel. Cette nouvelle fonctionnalité qui spécialise **Avatar** en fait d'ailleurs une application de RV très attrayante.

Les divers objectifs déjà mentionnés au sujet d'**Avatar** et déjà atteints avec la première version ont évidemment tous été respectés dans cette nouvelle version de l'application. Ainsi, il est par exemple toujours possible d'utiliser **Avatar** sur à peu près n'importe quel ordinateur. En effet, la nouvelle fonctionnalité introduite dans cette nouvelle version d'**Avatar** ne requiert qu'une simple caméra vidéo numérique, un périphérique désormais très répandu au niveau des ordinateurs personnels. De plus, cette fonctionnalité est optionnelle et il est évidemment toujours possible d'utiliser l'application telle que présentée au chapitre précédent.

La possibilité de contrôler les acteurs virtuels à partir d'acteurs réels a été élaborée en utilisant les fonctions offertes dans *ARToolKit*. Bien que les résultats d'estimation de la pose fournis par les algorithmes de cette boîte à outils soient généralement très satisfaisants, cette solution n'est pas parfaite et certains problèmes sont intrinsèquement liés à son utilisation.

En particulier, il peut être difficile pour une personne ayant peu de connaissance en mathématiques de créer les fichiers utilisés par *ARToolKit* et décrivant les marqueurs à reconnaître. En effet, certaines notions d'algèbre linéaire sont nécessaires afin d'être en mesure de décrire la pose d'un marqueur dans le repère de l'acteur. Cela va évidemment à l'encontre de l'objectif de convivialité et peut causer un problème au niveau de l'utilisation d'**Avatar** par des artistes n'ayant pas nécessairement de formation mathématique très poussée. Cette considération sera reprise en conclusion lorsque les améliorations possibles et les travaux futurs seront abordés.

Un autre problème lié à l'utilisation de *ARToolKit* pour estimer la pose des acteurs réels est que les algorithmes de vision numérique sont très sensibles aux conditions d'éclairage. L'ajout dans la barre d'outils d'**Avatar** de l'ascenseur permettant d'ajuster la valeur du seuil de segmentation vient en partie régler ce problème, mais cette dépendance à la lumière impose tout de même le respect de certaines conditions minimales d'éclairage. Cela est contradictoire avec l'utilisation d'un projecteur comme celui de la

salle de RV du LVSN, lequel requiert idéalement le moins de lumière ambiante possible. Cette difficulté peut cependant être surmontée en exploitant un éclairage propre à la maquette et n'émettant pas beaucoup de lumière ambiante à l'extérieur de celle-ci. Cela est d'ailleurs le cas du castelet réel pourvu de son propre éclairage à base de fibres optiques.

Ainsi, avec cette nouvelle fonctionnalité, **Avatar** a atteint la plupart des objectifs énoncés dans la définition de notre projet de maîtrise. En fait, un seul de ces objectifs de départ n'a pas encore été abordé, à savoir celui de la collaboration entre divers metteurs en scène physiquement distants les uns des autres. Le prochain chapitre aborde cette question.

Chapitre 4

Fonctionnalité de collaboration à distance par Avatar

4.1 Définition des objectifs

De tous les objectifs initiaux établis au début de notre projet de maîtrise, que ce soit les objectifs imposés par le projet de Castelet Électronique ou encore ceux que nous nous sommes personnellement fixés pour le développement d'**Avatar**, un seul d'entre eux n'a pas encore été abordé. En effet, dans la définition du projet de Castelet Électronique, l'application de RV servant à visualiser le castelet virtuel devait permettre la collaboration entre divers metteurs en scène. Ce dernier objectif encore non-abordé fait l'objet de ce chapitre.

4.1.1 Objectifs du Castelet Électronique

Lors de son élaboration au LANTISS, le Castelet Électronique a été défini comme un outil qui allait permettre à plusieurs metteurs en scène de collaborer à distance et en temps réel. Dans ce contexte, un même castelet virtuel allait être partagé par différents metteurs en scène. Chacun de ceux-ci allait en fait posséder une instance de ce (seul) castelet virtuel. Ces différentes instances allaient donc être interconnectées et les modifications apportées par l'un des metteurs en scène allaient être instantanément appliquées aux copies du castelet virtuel des autres metteurs en scène. Ainsi, selon cette définition, l'application permettant de visualiser le castelet virtuel allait devoir se brancher avec elle-même.

Cette dernière fonctionnalité allait être réalisée grâce aux possibilités offertes par le réseau Internet. Ce réseau est en effet idéal pour cette tâche puisqu'il est déjà implanté à la grandeur de la planète. Ainsi aucune infrastructure particulière ne devait être élaborée et cette nouvelle fonctionnalité allait pouvoir être utilisée par quiconque ayant un accès à Internet.

Cependant, il serait malheureux que cette nouvelle fonctionnalité vienne en annuler une autre déjà existante. Cette affirmation peut sembler triviale, mais c'est pourtant ce qui risquait d'arriver avec l'ajout de cette possibilité au Castelet Électronique.

En effet, supposons un instant que 2 metteurs en scène travaillent simultanément sur une même scène en partageant un seul et unique castelet virtuel dont chacun possède une copie. Supposons de plus que chacun de ces metteurs en scène dispose de son propre castelet réel pour interagir avec cet unique castelet virtuel. Lorsque le premier metteur en scène utilise son castelet réel pour contrôler le castelet virtuel, les acteurs du castelet virtuel se voient attribuer exactement la même pose que les acteurs de son castelet réel. Or, cette affirmation est aussi vraie pour le deuxième metteur en scène. Les acteurs virtuels sont donc maintenant chacun associés à 2 acteurs réels différents.

Le problème est que ces 2 acteurs réels n'ont pas nécessairement la même pose dans leur castelet réel respectif. En effet, lors du processus de mise en scène, les metteurs en scène modifient la pose de ces acteurs de façon indépendante.

Laquelle de ces 2 poses différentes l'acteur virtuel doit-il alors se voir attribuer ? L'atteinte du dernier grand objectif du projet de Castelet Électronique nécessite de trouver une solution à ce problème sans quoi cette nouvelle possibilité ne serait pas compatible avec l'utilisation d'acteurs réels.

4.1.2 Objectifs pour Avatar

Puisque le castelet virtuel est en fait visualisé avec **Avatar**, les objectifs, fonctionnalités et problèmes mentionnés ci-dessus au sujet du Castelet Électronique peuvent tous être considérés du point de vue de cette application de RV.

Ainsi, **Avatar** devra être une fois de plus modifié afin de répondre à ces exigences du projet de Castelet Électronique. **Avatar** devra donc dorénavant permettre à divers utilisateurs de partager un même monde virtuel simultanément.

De plus, chacun de ces utilisateurs devra être en mesure de modifier ce monde virtuel

partagé au même titre qu'il lui est actuellement possible de modifier un monde virtuel non partagé.

Les modifications apportées à un monde virtuel par un utilisateur devront aussi être immédiatement reflétées chez les autres utilisateurs partageant le même monde virtuel. Les diverses instances d'**Avatar** partageant un même monde virtuel devront donc être synchronisées.

Finalement, cette nouvelle fonctionnalité d'**Avatar** ne devra pas priver les utilisateurs de la possibilité d'utiliser des acteurs réels afin de contrôler les acteurs virtuels. Tel que soulevé précédemment, cela engendre cependant un problème sérieux auquel il faudra trouver une solution.

4.2 Solution proposée

L'implantation de cette nouvelle fonctionnalité d'**Avatar** peut être divisée en 2 parties. D'abord il faut permettre le partage d'un même monde virtuel entre différentes instances d'**Avatar**, puis il faut s'assurer que cela n'est pas incompatible avec l'utilisation d'acteurs réels.

4.2.1 Le partage de mondes virtuels

L'architecture utilisée afin de permettre le partage de mondes virtuels entre différentes instances d'**Avatar** est basée sur le modèle *client/serveur*. Ainsi, lorsque plusieurs utilisateurs décident de partager un même monde virtuel dans **Avatar**, l'un de ceux-ci doit d'abord être désigné comme le *serveur*. C'est celui-ci qui créera le monde virtuel qui sera ensuite partagé avec les autres utilisateurs, appelés *clients*.

Ces clients devront donc ensuite se connecter au serveur via Internet. Une fois la connexion établie, la description du monde virtuel créé par le serveur sera envoyée aux clients. Chacun des utilisateurs d'**Avatar** impliqué dans cette collaboration détiendra donc à ce moment une instance du même monde virtuel, peu importe qu'il s'agisse du serveur ou d'un client.

Comme nous l'avons vu, certains problèmes apparaissent lorsque l'on considère de permettre à plusieurs utilisateurs partageant un même monde virtuel de contrôler celui-

ci par le biais d'acteurs réels. Supposons donc pour l'instant qu'aucun des utilisateurs impliqués dans la collaboration n'utilise d'acteurs réels pour contrôler le monde virtuel partagé.

Afin d'assurer la synchronisation, chacune des modifications apportées (par le biais de la souris) au monde virtuel par le serveur est instantanément envoyée aux différents clients et **Avatar** modifie alors automatiquement leur instance du monde virtuel en conséquence.

De plus, chacune des modifications apportées (toujours par le biais de la souris) au monde virtuel par un client est instantanément envoyée au serveur. À ce moment, le serveur modifie son instance du monde virtuel et envoie les modifications aux autres clients.

De cette façon, dès qu'un utilisateur modifie sa copie du monde virtuel, la modification est instantanément appliquée à toutes les instances de ce monde virtuel reliées par le réseau. Ainsi, tel qu'il était souhaité, toutes les instances d'**Avatar** partageant un même monde virtuel sont toujours synchronisées.

Une chose à noter cependant est que la pose de la caméra virtuelle d'une instance particulière d'**Avatar** n'a rien à voir avec le contenu du monde virtuel qu'elle permet de visualiser. Ainsi, les modifications apportées à la pose de la caméra de l'instance particulière d'**Avatar** d'un certain utilisateur ne sont pas transmises aux autres utilisateurs.

Cependant, lors de la création d'un monde virtuel dans **Avatar**, la caméra virtuelle est initialement positionnée de sorte que l'origine du monde virtuel soit son point d'intérêt[1]. C'est d'ailleurs à cet endroit que les nouveaux acteurs sont ajoutés dans le monde virtuel. Cependant, rien n'empêche l'utilisateur de déplacer par la suite ces acteurs et de les positionner à un endroit non couvert par le champ de vue initial de cette caméra. Le monde virtuel partagé qui a initialement été élaboré du côté serveur pourrait à la limite être totalement en dehors du champ de vue initial de la caméra. Ainsi, afin de s'assurer que les clients puissent visualiser le monde virtuel dont ils reçoivent la description, la pose de la caméra virtuelle du serveur est envoyée avec la description du monde virtuel partagé lors de la connexion des clients. Cette pose est alors attribuée à la caméra virtuelle des clients, mais ces derniers ont toujours la possibilité de la modifier à leur guise par la suite.

Finalement, soulignons simplement que les clients peuvent à tout moment reposition-

[1]La pose initiale de la caméra virtuelle d'**Avatar** correspond en fait au point de vue prédéfini *perspective* accessible par le menu *Interaction*

ner leur caméra virtuelle exactement au même endroit que celle du serveur en utilisant le nouveau point de vue prédéfini à cet effet dans le menu *Interaction* d'**Avatar**.

4.2.2 La compatibilité avec l'utilisation d'acteurs réels

Avatar permet donc maintenant à plusieurs utilisateurs de se connecter ensemble par l'entremise du réseau Internet et de partager un même monde virtuel. Cela était bien notre objectif. Cependant, le travail n'est pas terminé puisqu'il faut maintenant s'assurer que cette nouvelle fonctionnalité n'entre pas en conflit avec l'utilisation d'acteurs réels.

Comme nous l'avons vu, ces deux fonctionnalités d'**Avatar** semblent a priori plutôt incompatibles. Cependant, ces fonctionnalités constituaient des exigences du projet de Castelet Électronique. Une solution devait donc être trouvée afin de permettre leur utilisation simultanée.

Supposons donc un instant que plusieurs utilisateurs d'**Avatar** désirent partager un même monde virtuel. Un de ceux-ci se désigne donc comme serveur et les autres s'y connectent en tant que clients. Si personne n'utilise d'acteurs réels, tout fonctionne parfaitement comme nous l'avons déjà vu. Maintenant, supposons que le serveur décide d'utiliser des acteurs réels pour contrôler les acteurs du monde virtuel.

De son côté, cela ne cause aucun problème. En effet, qu'il utilise des acteurs réels ou encore sa souris pour déplacer les acteurs virtuels ne change absolument rien pour le serveur. Cela ne change rien pour les communications avec les clients non plus. Les modifications apportées au monde virtuel par le biais des acteurs réels sont simplement envoyées aux clients tel qu'il est le cas lorsque ces modifications son apportées par le biais de la souris.

Le problème se présente cependant lorsqu'un client désire modifier le monde virtuel. En effet, voyons ce qui se produit lorsqu'un client modifie son instance du monde virtuel en déplaçant par exemple l'un des acteurs virtuels (par le biais de sa souris). Tel que mentionné précédemment, les modifications qu'il apporte sont premièrement transmises au serveur. Le serveur modifie alors son instance du monde virtuel et envoie la modification aux autres clients.

Cependant, du côté du serveur, l'acteur réel correspondant à l'acteur virtuel que le client a déplacé ne se déplace pas tout seul. Sa pose demeure donc inchangée dans le

monde réel. Ainsi, une fraction de seconde[2] après cette modification, la pose de cet acteur réel est recalculée et réaffectée à l'acteur virtuel correspondant, lequel reprend alors la pose qu'il avait juste avant la modification apportée par le client. Cette modification est ensuite transmise à chacun des clients comme c'est le cas pour toute modification, ce qui fait en sorte que l'effet net de l'action du client est nul. De cette façon, toute modification apportée au monde virtuel par un client est quasi immédiatement annulée par le serveur et les clients ne sont alors rien d'autre que des témoins des actions posées par le serveur au moyen de ses acteurs réels.

Pour remédier à cette situation, **Avatar** a été donc modifié de sorte que chaque acteur réel soit maintenant associé à 2 acteurs virtuels (identiques) plutôt qu'à un seul du côté serveur. Le premier de ces 2 acteurs, appelé *avatar pseudo-virtuel* a toujours une pose identique à l'acteur réel, peu importe les modifications apportées à sa pose par les clients. En fait, ces modifications sont maintenant plutôt apportées à la pose de l'autre acteur virtuel, appelé *avatar purement virtuel*, lequel a toujours une pose identique à celle de l'acteur virtuel correspondant des clients.

De cette façon, les modifications apportées au monde virtuel par les clients ne sont plus immédiatement annulées par le serveur. En effet, la pose de l'acteur réel estimée par *ARToolKit* n'est alors affectée qu'à l'avatar pseudo-virtuel dans l'instance du monde virtuel du serveur. Seules les modifications apportées à l'avatar purement virtuel du serveur sont transmises aux clients. Ainsi, du côté serveur, c'est comme si le monde réel et la composante pseudo-virtuelle du monde virtuel étaient toujours en adéquation, mais que ceux-ci étaient complètement dissociés de la composante purement virtuelle, laquelle correspond en fait au monde partagé avec les clients.

Les modifications apportées (par le biais de la souris) à la pose d'un acteur virtuel par un client sont donc apportées à la pose de l'avatar purement virtuel du serveur et transmises aux autres clients alors que la copie pseudo-virtuelle de cet acteur conserve sa pose (fournie par *ARToolKit* via la caméra).

Ensuite, le serveur peut déplacer l'acteur réel correspondant jusqu'à ce que l'avatar pseudo-virtuel (et donc par le fait même l'acteur réel) ait la même pose que l'avatar purement virtuel et donc que toutes les instances, autant réelle que virtuelle, du monde partagé concordent adéquatement. Après tout, l'intérêt d'utiliser des acteurs réels est de contrôler les acteurs du monde virtuel et non d'avoir deux mondes complètement dissociés.

[2] L'intervalle de temps exact dépend en fait du taux de rafraîchissement de la caméra utilisée mais peut être estimé à environ $\frac{1}{30}$ seconde.

Le problème qui se pose alors est qu'il est pratiquement impossible de donner à l'acteur réel exactement la même pose que celle de l'avatar purement virtuel. Le monde réel du serveur ne peut jamais être exactement identique au monde virtuel partagé. De plus, du côté du serveur, l'instance de ce monde virtuel partagé contient toujours 2 avatars virtuels de chaque acteur et ce dans des poses très similaires, mais non identiques, ce qui entraîne une incohérence inacceptable.

Ce problème a été résolu en donnant alors la possibilité au serveur de *fusionner* les 2 avatars virtuels lorsqu'il considère que la pose de l'avatar pseudo-virtuel (correspondant à la pose de l'acteur réel fournie par *ARToolKit*) est suffisamment semblable à celle de l'avatar purement virtuel (laquelle a été modifiée par un des clients). Cette *fusion* ajuste en fait la pose de l'avatar purement virtuel afin de lui donner exactement la même pose que l'avatar pseudo-virtuel (laquelle n'est pas modifiable puisque calculée par *ARToolKit*). Cet ajustement de la pose de l'avatar purement virtuel est alors ensuite immédiatement transmis aux clients et la cohérence entre le monde réel et le monde virtuel partagé est alors atteinte.

Suite à cela, les deux avatars sont fusionnés et tout déplacement de l'acteur réel induit alors une modification de la pose de ces deux avatars fusionnés et cette modification est envoyée aux clients (comme toute modification apportée à un avatar purement virtuel). Le monde réel et le monde virtuel partagé sont donc ainsi en adéquation constante jusqu'à ce qu'un des clients apporte une nouvelle modification au monde virtuel (par le biais de la souris).

Le partage d'un même monde virtuel entre plusieurs utilisateurs est ainsi possible même lorsque des acteurs réels sont utilisés par le serveur.

Le problème est qu'il est pratiquement impossible même avec cette méthode que deux (ou plus) castelets réels soient parfaitement identiques. Ainsi, il ne serait pas possible que plus d'un utilisateur emploie des acteurs réels pour contrôler le monde virtuel.

Cette problématique a été soulevée au LANTISS lors de l'élaboration du projet de Castelet Électronique. Afin d'y remédier, il a été convenu qu'un seul castelet réel allait être utilisé lors de mises en scène collaboratives. Cette idée est d'autant plus convenable que le castelet réel du LANTISS est une maquette spécialisée et unique en son genre offrant des possibilités hors du commun (entre autres grâce à son plancher robotisé).

C'est donc cette solution qui a été implantée dans **Avatar** : seul l'usager du côté serveur peut utiliser des acteurs réels pour contrôler le monde virtuel lorsque celui-ci

est partagé avec d'autres utilisateurs (clients). La section suivante présente un exemple d'exploitation de ces nouvelles fonctionnalités d'**Avatar**.

4.3 Présentation d'Avatar, l'application de réalité virtuelle collée à la réalité et permettant la collaboration

La poursuite de voitures élaborée au chapitre précédent sera maintenant considérée du point de vue d'une mise en scène collaborative afin d'illustrer cette nouvelle fonctionnalité d'**Avatar**.

Continuons donc la mise en scène là où nous l'avions laissée. La figure 4.1 présente l'état des castelets tels qu'ils étaient à la fin de l'exemple du chapitre précédent. Comme au chapitre précédent, nous utiliserons la maquette présentée à la figure 4.1(b) pour faire cette mise en scène.

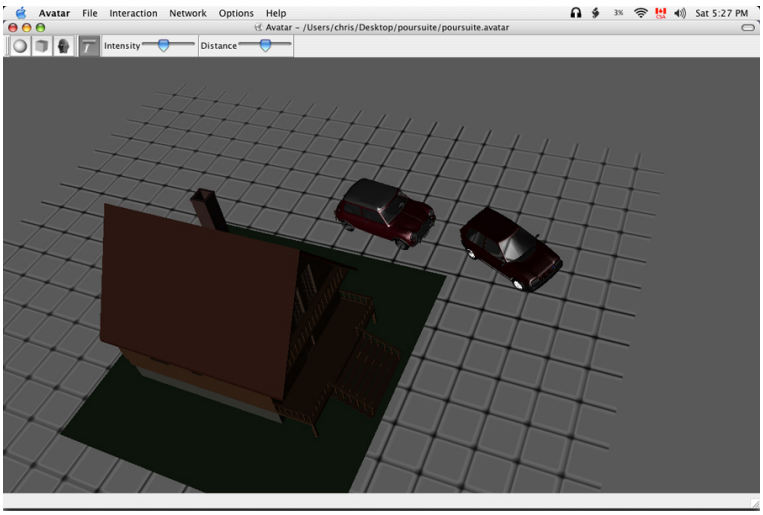
Supposons maintenant que l'on désire collaborer avec 2 autres metteurs en scène situés à distance. Comme nous possédons le castelet réel et que nous avons d'ailleurs déjà commencé la mise en scène, nous agirons à titre de serveur et nous partagerons le monde virtuel déjà élaboré avec nos collaborateurs. Notons qu'il n'y a qu'une seule version d'**Avatar** et que celle-ci peut être utilisée autant comme serveur que comme client (évidemment une seule de ces deux options est disponible à la fois).

Avatar possède maintenant un nouveau menu intitulé *Network* (voir figure 4.2).

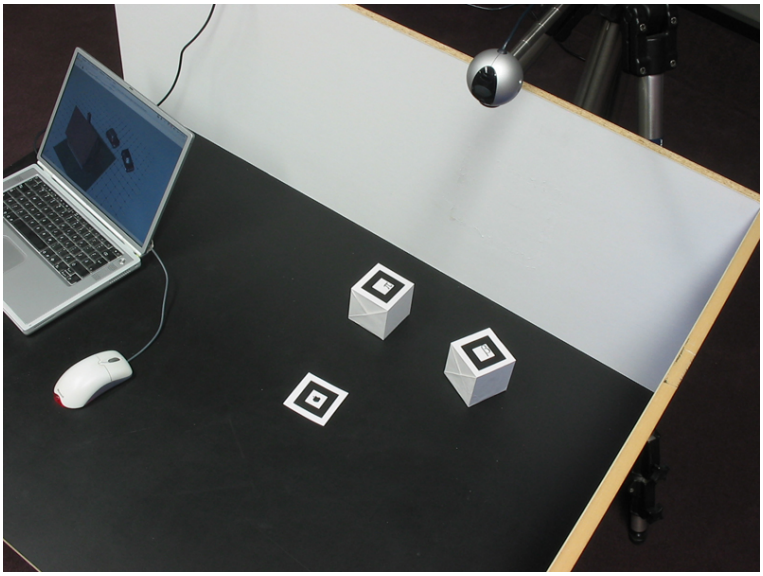
Tout ce que nous avons à faire est de sélectionner la fonction *Start Server* de ce menu et de donner notre adresse IP à nos collaborateurs. Ceux-ci n'ont alors qu'à sélectionner la fonction *Connect to Server* du même menu et à entrer cette adresse IP afin de se connecter au serveur et ainsi recevoir la description de notre monde virtuel maintenant partagé. La figure 4.3 illustre ces propos.

À ce moment, les 3 instances du castelet virtuel sont en parfaite adéquation avec le castelet réel. De plus, les caméras virtuelles des collaborateurs (clients) ont exactement la même pose que celle du serveur.

À partir de ce moment, notre monde virtuel est entièrement partagé avec nos 2 collaborateurs et ceux-ci peuvent alors lui apporter toutes les modifications qu'ils dé-



(a) État du castelet virtuel.



(b) État du castelet réel utilisé pour contrôler le castelet virtuel.

FIG. 4.1 – État actuel des castelets pour la mise en scène de la poursuite de voitures.

sirent, exactement au même titre que nous (à l'exception qu'ils ne peuvent se servir d'acteurs réels pour apporter ces modifications). Ils peuvent alors ajouter, déplacer ou encore supprimer des acteurs comme ils le désirent et ces modifications seront apportées à toutes les instances du monde virtuel. De la même façon, toutes les modifications que nous apportons au monde virtuel leur sont instantanément transmises et appliquées à leur instance du castelet virtuel.

Supposons donc qu'un de nos collaborateurs (un client) ajoute un nouvel acteur à la

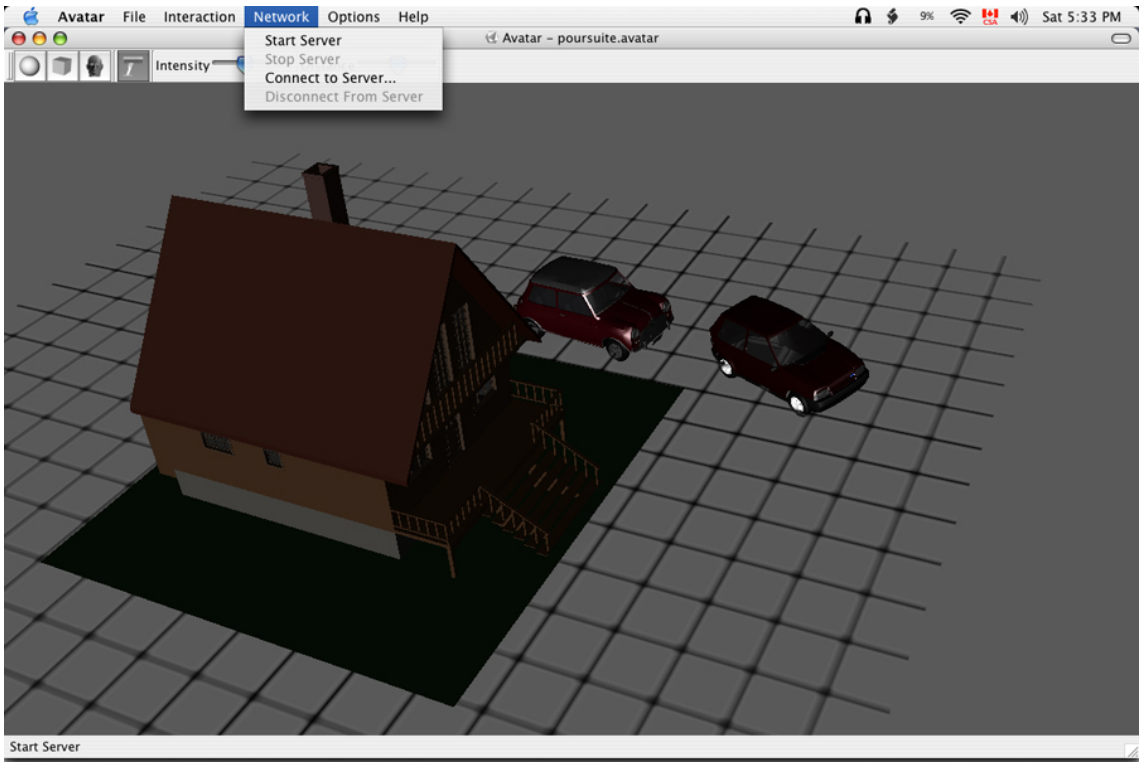


FIG. 4.2 – Le menu *Network* d'**Avatar**.

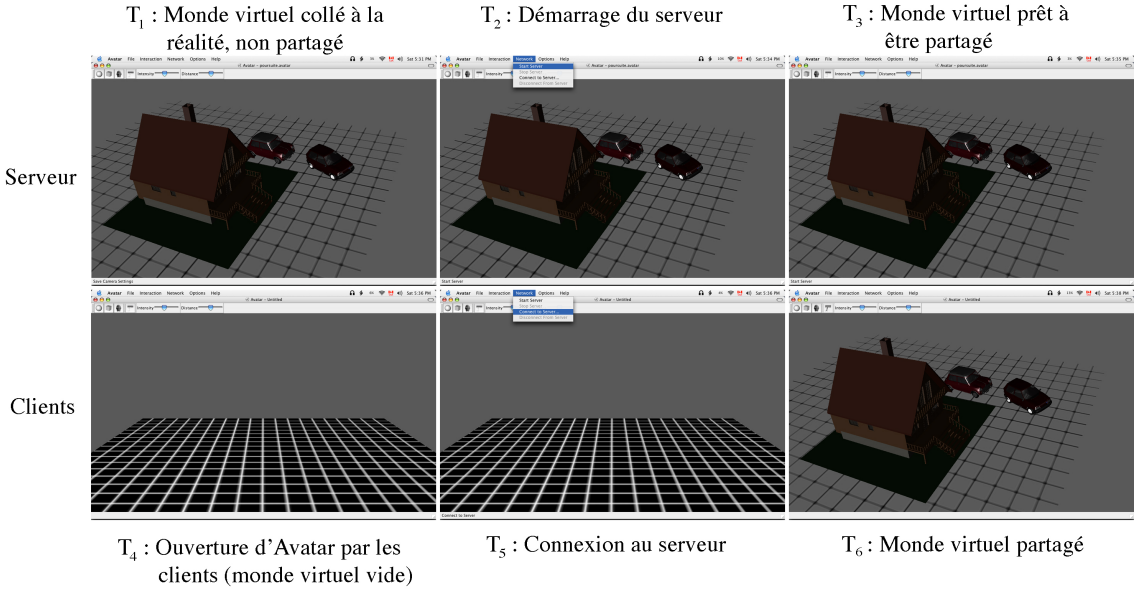


FIG. 4.3 – Partage d'un monde virtuel dans **Avatar**.

scène. Puisque nous sommes le serveur, cette modification nous est alors immédiatement transmise et nous la transmettons aussitôt à l'autre collaborateur avec un avertissement sur l'ajout d'un acteur (voir figure 4.4).

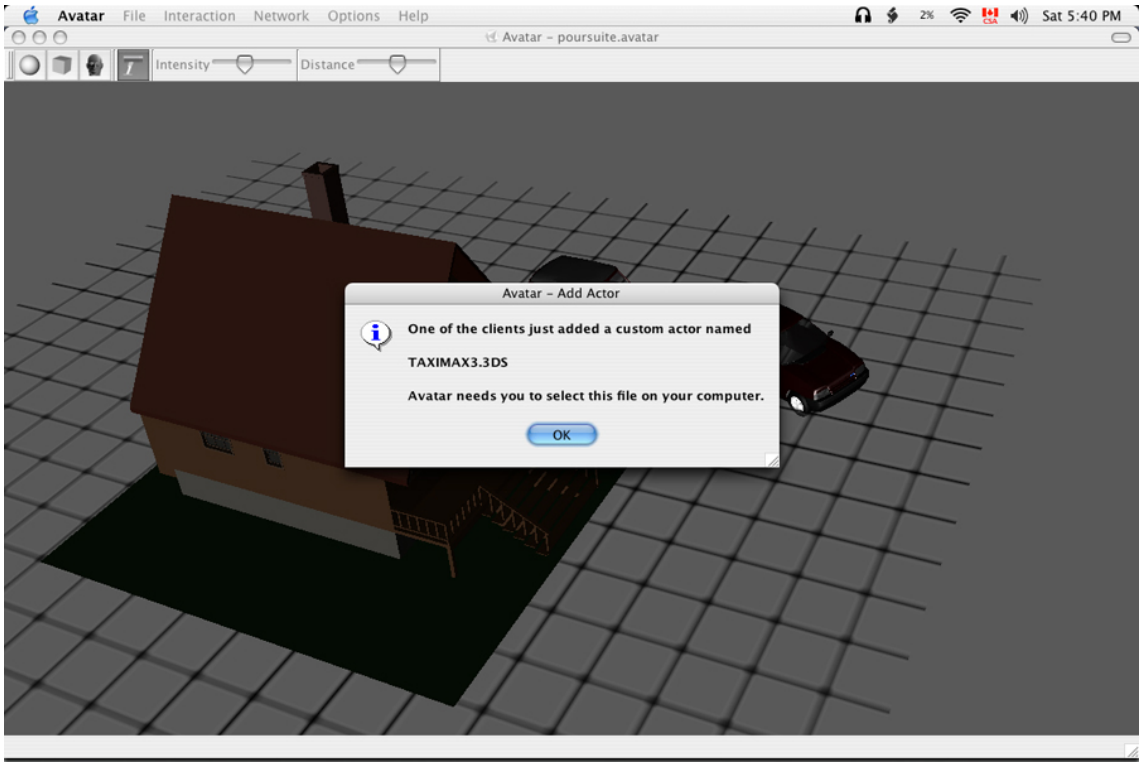
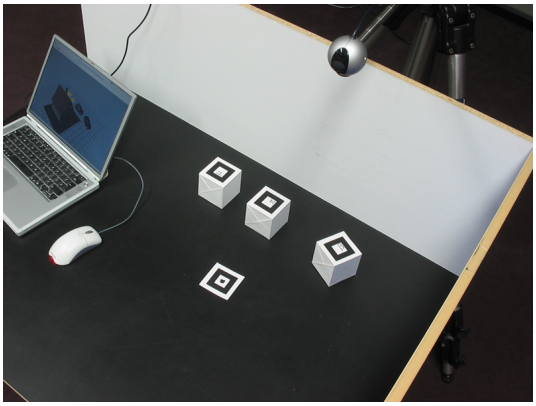


FIG. 4.4 – **Avatar** prévient l'utilisateur lorsqu'un collaborateur ajoute un acteur au monde virtuel partagé.

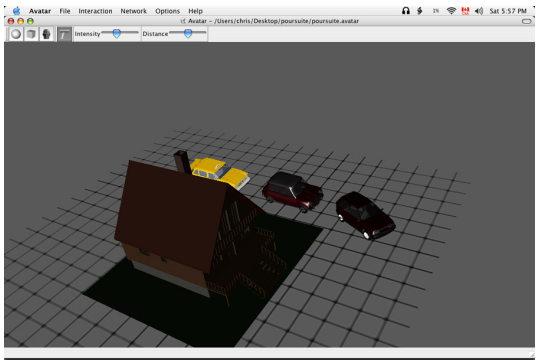
Si ce nouvel acteur ajouté par un collaborateur n'est pas l'un des acteurs prédéfinis dans **Avatar**, l'application nous demande alors de lui indiquer où est situé sur notre ordinateur[3] le fichier (de format tiff, 3ds ou wrl) décrivant cet acteur.

Suite à cela, puisque nous sommes le serveur et avons la possibilité d'utiliser des acteurs réels pour contrôler le monde virtuel, **Avatar** nous demande si ce nouvel acteur est associé à un acteur dans le monde réel et, le cas échéant, de lui indiquer quel fichier décrit cet acteur. Dans notre exemple, ce nouvel acteur est bien associé à un acteur réel (que nous avons au préalable décrit dans un fichier situé sur notre ordinateur). Nous informons donc **Avatar** du fichier décrivant cet acteur réel et ajoutons alors celui-ci à la scène réelle. Notons qu'**Avatar** ne demande pas ces informations à l'autre collaborateur puisque celui-ci agit à titre de client et n'a donc pas la possibilité d'utiliser des acteurs réels.

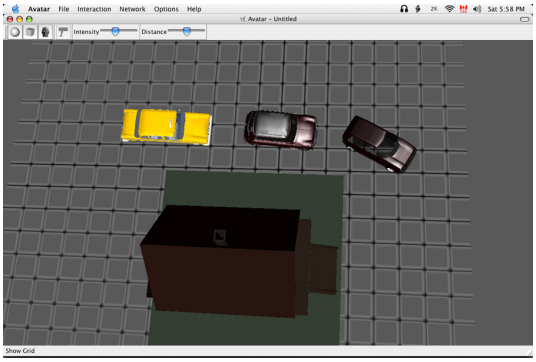
[3]Présentement, aucun transfert de fichier n'est disponible à partir d'**Avatar** directement. Il serait cependant intéressant d'implanter cette fonctionnalité dans une future version de l'application.



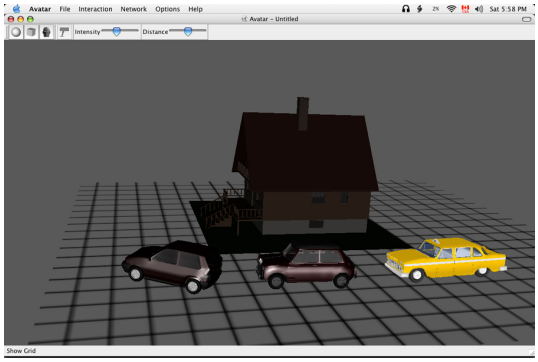
(a) Castelet réel.



(b) Serveur.



(c) Client 1.



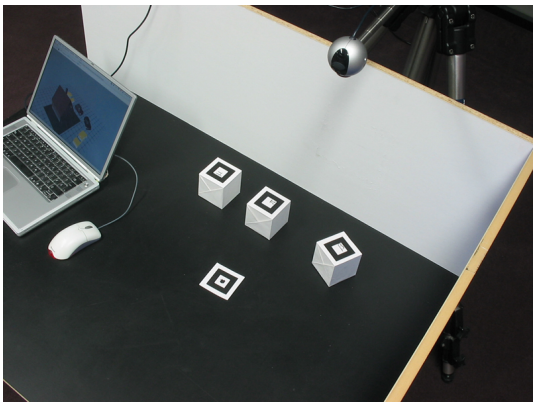
(d) Client 2.

FIG. 4.5 – État des castelets réel et virtuel après l'ajout d'un nouvel acteur par un collaborateur et son positionnement par *ARToolKit*.

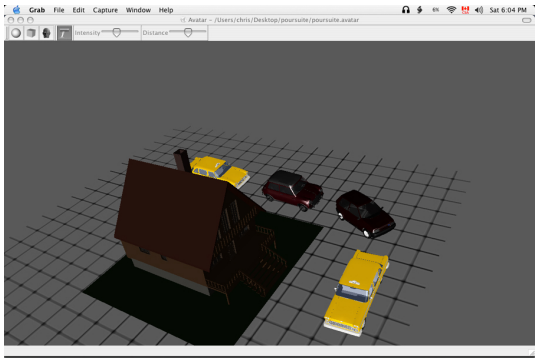
Une fois qu'il possède toute l'information nécessaire, **Avatar** ajoute le nouvel acteur à la scène virtuelle et le positionne selon la pose fournie par *ARToolKit*. Nous pouvons maintenant déplacer le nouvel acteur virtuel par le biais de sa contrepartie réelle sur le castelet réel et ces déplacements sont alors transmis et appliqués aux acteurs correspondants dans les instances du castelet virtuel de nos collaborateurs. La figure 4.5 présente l'état des castelets réel et virtuel après l'ajout du nouvel acteur et son positionnement par *ARToolKit*.

Tel qu'on peut le voir sur cette figure, bien que les instances du castelet virtuel de nos collaborateurs soient exactement identiques à la nôtre, leur point de vue sur ce monde virtuel peut être complètement différent du nôtre. En effet, tel que mentionné précédemment, les modifications apportées à la pose de la caméra ne sont pas transmises sur le réseau d'**Avatar** et la pose de la caméra virtuelle d'une instance particulière d'**Avatar** est totalement indépendante de la pose de la caméra virtuelle des autres instances partageant le même monde virtuel.

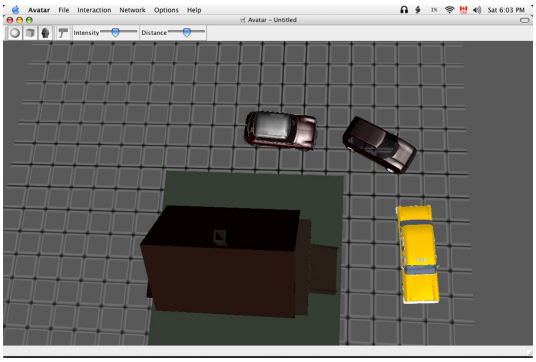
Supposons maintenant qu'un de nos collaborateurs déplace (avec sa souris) l'un des acteurs virtuels de la scène. De son côté, tout se passe normalement et l'acteur est déplacé exactement comme il le serait dans le cas où le monde virtuel ne serait pas partagé. Cependant, cette modification n'est pas sans conséquences. En fait, celle-ci nous est transmise et nous la transmettons aussitôt à l'autre client qui voit alors sa copie de cet acteur se déplacer jusqu'à l'endroit décidé par l'autre collaborateur. De notre côté, puisque nous utilisons des acteurs réels[4], **Avatar** applique cette modification à l'avatar purement virtuel seulement. À ce moment, les 2 copies de l'acteur (l'avatar purement virtuel et l'avatar pseudo-virtuel) se séparent et celui-ci est donc présent 2 fois dans notre instance du castelet virtuel tel qu'illustré à la figure 4.6.



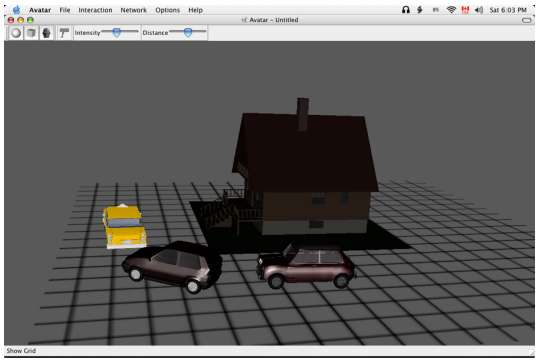
(a) Castelet réel.



(b) Serveur.



(c) Client 1.



(d) Client 2.

FIG. 4.6 – État des castelets réel et virtuel après qu'un des clients ait déplacé un acteur virtuel. Puisque la nouvelle pose de cet acteur virtuel diffère de celle de son homologue réel, l'avatar purement virtuel s'est séparé de l'acteur pseudo-virtuel chez le serveur.

Afin de rétablir la cohérence entre le monde réel et le monde virtuel, nous déplaçons alors l'acteur réel correspondant jusqu'à ce qu'il ait la pose proposée par notre colla-

[4]Notons que si nous n'utilisions pas d'acteurs réels, la modification serait appliquée à notre acteur virtuel exactement comme c'est le cas pour l'autre client.

borateur, c'est-à-dire jusqu'à ce que la pose de l'avatar pseudo-virtuel (et donc celle de l'acteur réel) concorde avec celle de l'avatar purement virtuel (identique à celle de l'acteur virtuel dans les instances du monde virtuel des clients). Lorsque ces 2 avatars ont une pose suffisamment semblable, **Avatar** nous prévient de la situation. En effet, lorsque la différence entre la pose des 2 avatars est suffisamment petite[5], **Avatar** met en évidence l'avatar pseudo-virtuel tel que présenté à la figure 4.7. À ce moment, nous fusionnons les 2 avatars afin de réajuster la pose de l'avatar purement virtuel et donc celle de l'instanciation de cet acteur chez nos collaborateurs (voir figure 4.8).

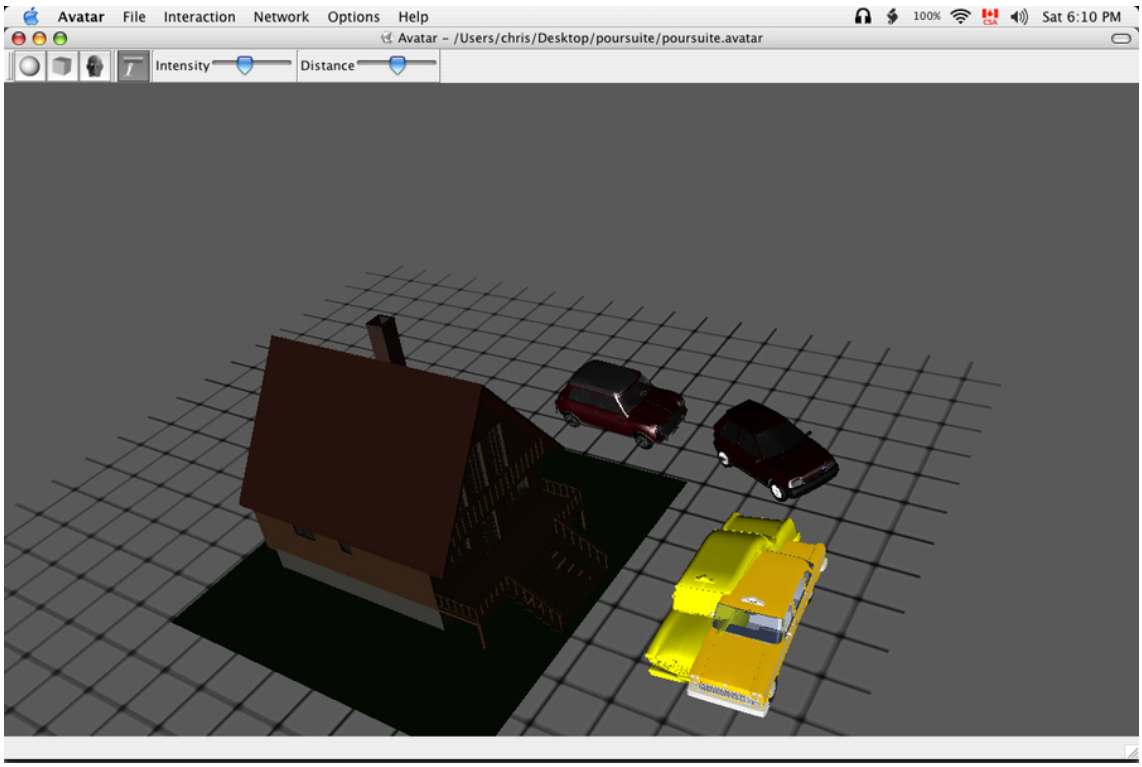


FIG. 4.7 – **Avatar** met en évidence l'avatar pseudo-virtuel pour indiquer à l'utilisateur que la différence entre la pose des 2 avatars est suffisamment petite.

Notons que nous pouvons fusionner les 2 avatars à n'importe quel moment sans que leur pose ait à être semblable. Ainsi, si nous n'aimons pas la suggestion de notre collaborateur, nous pouvons immédiatement l'annuler sans avoir à d'abord la réaliser sur le castelet réel.

De plus, la barre d'outils d'**Avatar** contient maintenant un ascenseur permettant d'ajuster la valeur à partir de laquelle la différence entre les 2 poses est considérée négligeable par **Avatar**, c'est-à-dire la valeur à partir de laquelle il met en évidence

[5]Comme nous le verrons, le serveur a la possibilité d'ajuster le seuil à partir duquel la différence entre les deux poses est jugée négligeable.

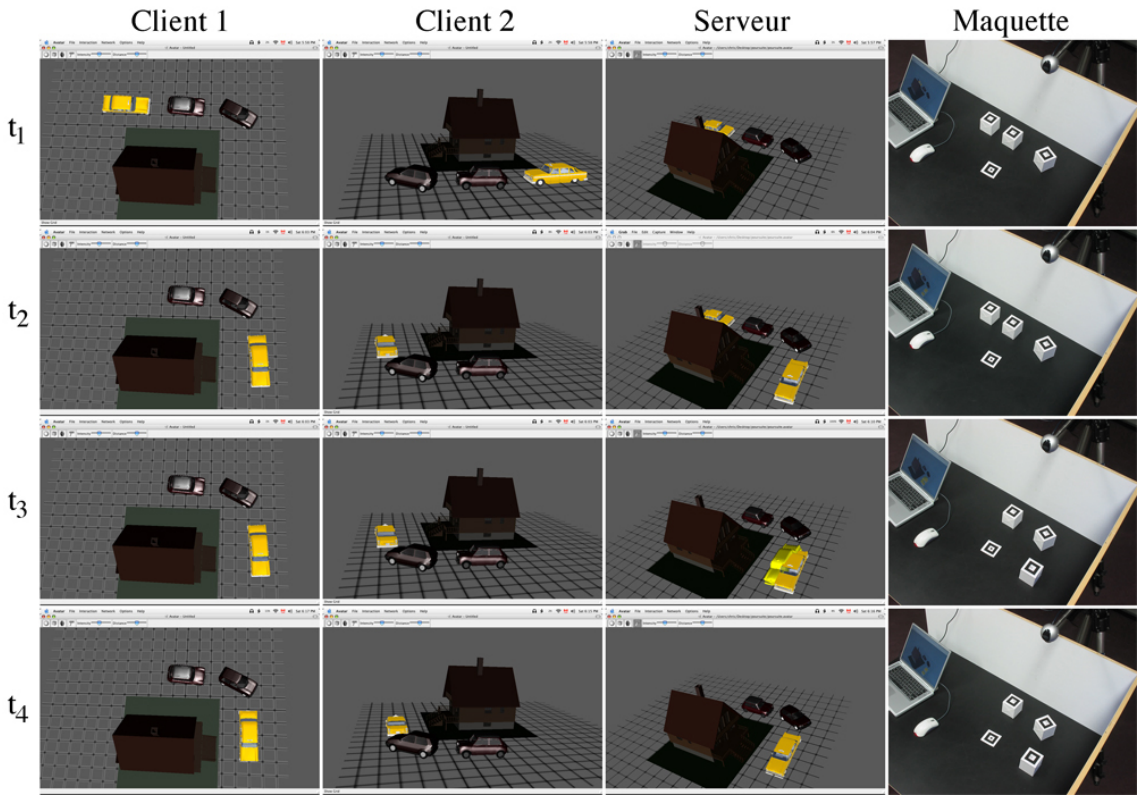


FIG. 4.8 – Exemple de conservation de l'adéquation entre le monde réel et le monde virtuel dans **Avatar** après une modification du monde virtuel par un collaborateur.

l'acteur pseudo-virtuel pour nous aviser que les 2 poses sont semblables. Cette option nous permet d'ajuster le degré de précision avec lequel nous voulons recréer sur le castelet réel les modifications apportées par nos collaborateurs dans le monde virtuel. Cette fonctionnalité pourrait aussi être plus utile dans le futur ; nous y reviendrons en conclusion lorsque nous aborderons les travaux futurs et les améliorations possibles.

Maintenant, les nouvelles fonctionnalités d'**Avatar** présentées dans ce chapitre peuvent aussi être utilisées dans un autre but que pour collaborer avec d'autres utilisateurs. En effet, il est possible de démarrer plusieurs instances d'**Avatar** sur un même ordinateur et de les interconnecter pour nous permettre de visualiser notre monde virtuel de plusieurs points de vue en même temps. Cette possibilité illustrée à la figure 4.9 est réalisée en démarrant d'abord un serveur puis en démarrant et connectant ensuite autant de clients que l'on désire sur celui-ci.

Cette façon de faire ressemble un peu à ce que l'on retrouve régulièrement dans les logiciels de modélisation et de rendu 3D. Bien que les fonctionnalités réseau d'**Avatar** n'aient pas été conçues pour cet usage, celui-ci est tout à fait possible et s'avère même très utile en pratique.

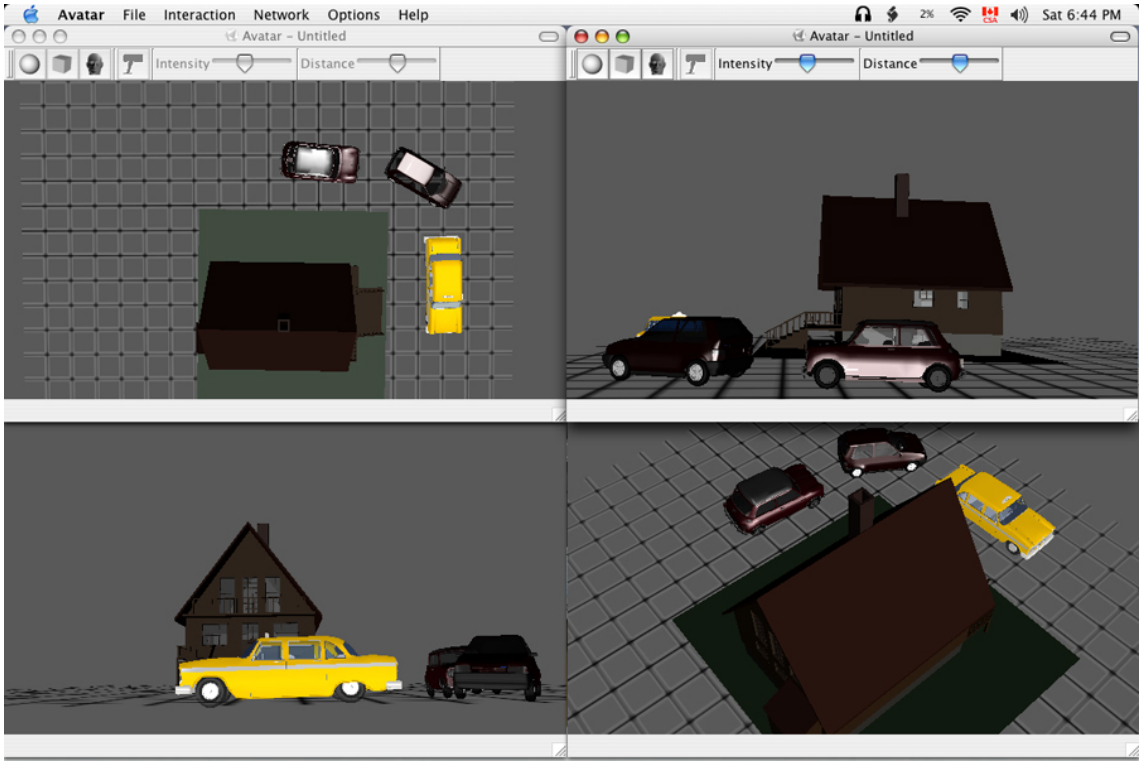


FIG. 4.9 – Plusieurs instances d'**Avatar** peuvent être démarrées sur le même ordinateur et interconnectées afin de présenter simultanément à l'utilisateur différents points de vue du monde virtuel.

4.4 Synthèse récapitulative

Avec ces nouvelles fonctionnalités, **Avatar** est maintenant une application de RV multi-utilisateurs à monde virtuel partagé. **Avatar** permet en effet à plusieurs utilisateurs d'évoluer dans un même monde virtuel et même que chacun puisse le modifier à sa guise. Cette opportunité leur est dans un premier temps disponible en réalité *purement* virtuelle, c'est-à-dire dans le cas où tous les utilisateurs interagissent avec le monde virtuel par le biais de leur souris (ou d'un autre périphérique de RV tel un gant de réalité virtuelle (voir figure 2.1)).

De plus, afin de respecter les objectifs du projet de Castelet Électronique, cette nouvelle fonctionnalité peut aussi être utilisée de pair avec l'emploi d'acteurs réels (pour un seul utilisateur) afin de contrôler le monde virtuel.

Ces fonctionnalités constituaient les derniers objectifs du projet de Castelet Électronique qui n'avaient pas encore été abordés. Ainsi, tous les objectifs de ce projet d'envergure ont été atteints.

Évidemment, certaines fonctionnalités d'**Avatar** laissent cependant place à amélioration. Celles-ci seront d'ailleurs discutées en détail au chapitre suivant.

Chapitre 5

Conclusion

5.1 Retour sur les contributions

Durant ce travail, nous avons développé une application logicielle appelée **Avatar**. Il s'agit d'une application de RV conviviale, contrôlable par des acteurs réels et permettant la collaboration à distance. **Avatar** est en fait une application de RV exploitant des principes de vision par ordinateur afin d'offrir la possibilité à ses utilisateurs de contrôler les acteurs du monde virtuel par le biais d'acteurs réels. **Avatar** permet aussi à plusieurs utilisateurs de partager un même monde virtuel. Afin d'illustrer ces propos, la figure 5.1 résume quelques-unes des possibilités offertes par **Avatar**.

Avatar a été élaboré de sorte qu'il puisse être utilisé par des gens n'ayant pas nécessairement d'expertise dans le domaine. En fait, **Avatar** a d'abord et avant tout été développé comme un nouvel outil pour les metteurs en scène. Cependant notre application n'est pas limitée à ce domaine et peut servir à bien d'autres fins. Il s'agit en fait d'une application de RV au sens large offrant plusieurs possibilités uniques. **Avatar** peut en ce sens servir à toute application où la synchronisation entre la géométrie du monde réel et celle du monde virtuel est importante.

Bien qu'elle puisse être utilisée avec des équipements spécialisés de RV, cette application peut aussi très bien être utilisée sur tout ordinateur typique muni d'un clavier, d'une souris et optionnellement d'une caméra vidéo. **Avatar** peut ainsi être utilisée même pour des applications où le coût serait un problème.

Avatar est d'ailleurs une application gratuite disponible pour les principales plates-formes. Le code d'**Avatar** est en fait ouvert et librement accessible sur Internet [28]

FIG. 5.1 – **Avatar** est une application de RV offrant la possibilité à ses utilisateurs de contrôler des acteurs virtuels par le biais d'homologues réels et de partager un monde virtuel avec d'autres utilisateurs. Haut : Utilisateur *serveur* visualisant le castelet virtuel en stéréo avec des lunettes LCD de marque *CrystalEyes* et modifiant son contenu via le castelet réel dans la salle de réalité virtuelle du LVSN. Bas : Utilisateur *client* à distance visualisant le même castelet virtuel sur un écran avec des lunettes LCD de marque *CrystalEyes* pour une visualisation stéréoscopique et modifiant son contenu via sa souris et son clavier.

tout comme les versions compilées pour les différents systèmes d'exploitation. Cela offre la possibilité à quiconque d'adapter notre application à ses besoins spécifiques.

5.2 Exposition des principaux défis

Le premier et le plus grand défi à relever lors de la réalisation de ce projet était évidemment la conception et l'implantation d'une application de RV de toutes pièces à partir uniquement de quelques librairies de fonctions C++. Cette tâche a d'ailleurs occupé la plus grande partie du temps alloué au projet et ce, même en exploitant des outils simplifiant certaines parties du développement.

L'application de RV désirée devait en plus être conviviale et même être utilisable par des personnes n'ayant possiblement aucun bagage dans ce domaine. Cette contrainte n'a évidemment pas simplifié la programmation et a d'ailleurs eu un impact majeur sur le résultat final d'**Avatar**. En effet, l'intégration d'une IUG dans **Avatar** ne fut pas aussi simple que prévu. D'ailleurs, une première version de l'application fut complètement abandonnée après quelques mois de développement. L'architecture des classes fut alors entièrement repensée et modifiée afin de tenir compte des principes de fonctionnement de *Qt*.

Un autre défi important de ce projet était celui de permettre l'utilisation d'acteurs réels comme périphériques d'interaction avec les acteurs du monde virtuel. Il fallait dans ce cas assurer une adéquation constante entre le contenu du monde réel et celui du monde virtuel.

Cette adéquation fut dans un premier temps assurée en adaptant le contenu du monde virtuel de façon à ce qu'il corresponde toujours à celui du monde réel. Pour cela, il fallut être en mesure d'estimer la pose de chacun des acteurs présents dans le monde réel.

La détermination de la pose des acteurs réels et l'affectation de la même configuration aux acteurs virtuels constituaient donc la première partie du défi de l'adéquation des mondes réel et virtuel.

Ce défi fut relevé par l'utilisation de *ARToolKit*. Cependant, beaucoup de travail a dû être fait à ce sujet afin d'exploiter correctement les fonctions d'estimation de la pose de *ARToolKit* pour contrôler les acteurs virtuels créés et affichés à partir de VTK. D'ailleurs, d'après nos recherches sur Internet et en particulier sur les forums de

discussion présents sur les sites de *ARToolKit* [30] et VTK [23], l'utilisation simultanée de ces 2 outils constitue une contribution unique.

Un autre défi important de notre projet était de permettre la collaboration entre divers utilisateurs par le partage interactif d'un monde virtuel. Cela nécessita entre autres l'apprentissage de certaines notions au niveau des réseaux informatiques.

La plus grande difficulté liée au partage de mondes virtuels venait cependant du fait qu'**Avatar** permet l'utilisation d'acteurs réels pour contrôler les acteurs virtuels. Comme nous l'avons vu, cette problématique fût en partie réglée en imposant que l'utilisation d'acteurs réels ne soit possible que par un seul utilisateur à la fois.

Cependant, nous devions tout de même permettre l'interaction avec des acteurs virtuels directement dans l'environnement virtuel (par les collaborateurs) et assurer ensuite l'adéquation entre les mondes (réel et virtuel) en modifiant plutôt le monde réel de façon à ce qu'il concorde avec le monde virtuel partagé.

C'est ce qui constituait la deuxième partie du défi de l'adéquation des mondes réel et virtuel. Comme nous l'avons vu, ce défi fut relevé par l'utilisation de deux instanciations de chaque acteur virtuel du côté serveur.

5.3 Améliorations possibles et travaux futurs

Maintenant qu'**Avatar** peut être considéré fonctionnel, il serait intéressant de procéder à une validation de ce nouvel outil de mise en scène collaborative. Cette validation ainsi que les améliorations qu'il pourrait en résulter feront l'objet de travaux futurs. D'ailleurs, nous savons déjà qu'il y a encore place à amélioration dans **Avatar**.

Entre autres, nous prévoyons faire en sorte que les textures des modèles 3DS soient appliquées sur les modèles ajoutés aux mondes virtuels dans **Avatar**, ce qui n'est actuellement pas le cas. Cela augmenterait énormément le réalisme des acteurs virtuels. Cette amélioration devrait en fait être apportée au code de VTK et non à celui d'**Avatar** qui utilise ce code.

Dans la même lignée, il serait aussi intéressant de donner la possibilité d'inclure une carte d'environnement aux mondes virtuels d'**Avatar**. Cette nouvelle possibilité, qui simplifierait la création du décor et augmenterait de plus le réalisme, passe elle aussi par la modification du code de VTK.

Il n'y a présentement aucune rétroaction sensorielle dans **Avatar**. En effet, les déplacements de l'utilisateur ne sont actuellement pas pris en compte et la navigation dans les mondes virtuels d'**Avatar** est effectuée par le biais de la souris. Tel que mentionné cependant, **Avatar** a été codé de sorte qu'il serait très facile d'ajouter la possibilité d'utiliser un système de positionnement afin d'offrir une rétroaction sensorielle. Ce genre de système étant très dispendieux, son utilisation ne devait pas être obligatoire dans **Avatar**.

Cependant, la rétroaction sensorielle pourrait être réalisée à peu de frais. En effet, il est envisageable de fixer un marqueur sur une casquette ou un chapeau quelconque et d'utiliser *ARToolKit* pour estimer la pose de l'utilisateur. Celui-ci pourrait alors se déplacer comme il veut dans le monde virtuel en se déplaçant simplement de la même façon dans le monde réel. Toutefois, l'utilisation de *ARToolKit* est problématique dans un environnement avec peu de lumière ambiante comme dans une salle de RV. De plus, cette solution n'est pas vraiment appropriée pour gérer de grands déplacements de la part des usagers.

Une autre amélioration envisageable pour **Avatar** serait de considérer les sources de lumière comme des acteurs dans le monde virtuel. Il serait ainsi possible de créer des sources de lumière sur mesure et de facilement les positionner à l'endroit désiré à l'aide de la souris (ou encore par le biais d'un acteur réel). Il serait aussi intéressant pour les metteurs en scène de pouvoir modéliser les sources de lumière présentes sur le castelet réel afin de les intégrer au castelet virtuel. Les collaborateurs pourraient alors modifier celles-ci au même titre qu'il peuvent actuellement modifier les acteurs et **Avatar** deviendrait alors une plate-forme de prototypage collaboratif de l'éclairage de la scène.

Aussi, le concept de sélection n'est actuellement pas présent dans **Avatar**. Dans le mode *Actor* (c'est-à-dire lorsque les actions posées avec la souris sont appliquées aux acteurs et non à la caméra), les actions posées à partir de la souris sont effectuées sur l'acteur situé sous le pointeur de la souris et non sur un acteur préalablement sélectionné. Le concept de sélection pourrait certainement améliorer le caractère intuitif des diverses opérations disponibles dans **Avatar**.

À ce jour, **Avatar** ne détecte pas les collisions entre acteurs virtuels. De plus, le concept de gravité n'a pas été implanté. Ceci pourrait faire en sorte qu'un collaborateur propose une configuration du monde virtuel qu'il soit impossible de réaliser en pratique dans le monde réel. Il serait ainsi intéressant d'ajouter ces fonctionnalités à **Avatar** dans le futur, ce qui pourrait être accompli via un engin de simulation physique tel ODE [32].

Une fonctionnalité qu'il serait simple d'ajouter à **Avatar** serait la sauvegarde automatique de l'état du monde virtuel après chaque opération effectuée par un utilisateur. Cela permettrait d'offrir la fonction *Annuler* souvent utile dans les travaux de création et de modification comme celui de la mise en scène. Cependant, l'implantation d'une fonction *Annuler* sur le réseau n'est pas une tâche triviale et demanderait une certaine réflexion.

La convivialité de l'utilisation d'**Avatar** pourrait être améliorée en simplifiant la tâche de création des fichiers utilisés par *ARToolKit*. **Avatar** a été élaboré pour être exploité par des metteurs en scène. Or ces artistes ne possèdent pas nécessairement les notions d'algèbre linéaire requises pour élaborer le type de fichier utilisé par *ARToolKit*. À cet égard, des exemples de tels fichiers sont actuellement disponibles sur le site Internet d'**Avatar** [28]. Ces exemples sont standards et suffisants pour la plupart des cas. Cependant, afin de permettre aux usagers d'utiliser des configurations particulières, un programme muni d'une IUG pourrait être construit afin de combler cette lacune.

Au niveau de la mise en scène cinématographique (et au niveau d'une application de RV au sens large), il serait aussi intéressant qu'**Avatar** permette de faire suivre un parcours préprogrammé à la caméra. Cela permettrait de faire la mise en scène de séquences où la caméra est en mouvement. Cette possibilité aurait d'ailleurs été très intéressante lors de l'élaboration de la scène de combat présentée à la section 2.3. En effet, dans cette séquence, les acteurs sont statiques et la caméra se déplace autour de la scène.

Toujours dans la même lignée, il serait intéressant que les acteurs puissent être animés dans le monde virtuel. Ceci pourrait être réalisé par l'emploi d'un standard tel VRML. Ce standard est d'ailleurs déjà accepté par **Avatar**. Cependant, l'importateur des fichiers de ce type dans VTK ne considère qu'une partie de l'information qu'ils contiennent, à savoir la géométrie et la couleur des objets. L'information concernant les animations contenues dans ces fichiers n'est simplement pas considérée par VTK (tout comme l'information sur la texture et les matériaux). Ainsi, cette amélioration d'**Avatar** passe une fois de plus par la modification du code de VTK.

Actuellement, lorsque la pose de l'avatar pseudo-virtuel est suffisamment semblable à la pose de l'avatar purement virtuel, **Avatar** met en évidence l'avatar pseudo-virtuel pour informer l'utilisateur de la situation. Ces deux avatars sont ensuite fusionnés par l'utilisateur en cliquant sur l'un ou l'autre de ces acteurs virtuels avec le deuxième bouton de la souris alors que la touche *shift* est enfoncée. Il pourrait être intéressant de faire en sorte que les deux avatars se fusionnent automatiquement lorsqu'ils ont une pose suffisamment semblable. D'ailleurs, bien qu'elle soit déjà utile, c'est surtout pour

cette éventuelle fonctionnalité qu'un ascenseur permettant d'ajuster la valeur à partir de laquelle les deux poses sont considérées suffisamment semblables a été ajouté à la barre d'outils d'**Avatar**. La possibilité de fusionner à l'aide de la souris devra cependant être conservée puisque celle-ci permet d'annuler les modifications non désirées d'un collaborateur sans avoir à les réaliser préalablement avec les acteurs réels.

Finalement, une amélioration qui pourrait être apportée à **Avatar** dans l'optique de son utilisation par les metteurs en scène au LANTISS serait d'interfacer le plancher robotisé et le système d'éclairage à base de diodes électroluminescentes et de fibres optiques du castelet réel. Cela donnerait plus de contrôle aux collaborateurs qui pourraient alors configurer l'allure géométrique et l'éclairage de la scène à partir du castelet virtuel (visualisé dans **Avatar**). La collaboration entre les metteurs en scène ne serait plus alors uniquement au niveau du contenu (pose des acteurs), mais serait ainsi étendue à l'ensemble des paramètres de la mise en scène. Cette fonctionnalité sans précédent augmenterait encore plus les possibilités offertes aux metteurs en scène par cet outil unique qu'est le Castelet Électronique.

Bibliographie

[1] Laboratoire de Vision et Systèmes Numériques, Université Laval, Québec.
http://vision.gel.ulaval.ca/fr/

[2] Laboratoire de robotique de l'Université Laval, Université Laval, Québec.
http://wwwrobot.gmc.ulaval.ca/

[3] Centre d'optique, photonique et laser, Université Laval, Québec.
http://www.copl.ulaval.ca/

[4] Centre d'artistes Avatar, Coopérative Méduse, Québec.
http://www.meduse.org/avatar/

[5] Ex Machina, Québec.
http://www.exmachina.qc.ca/intro.htm

[6] POV-Ray, ThePersistence of Vision Raytracer.
http://www.povray.org/

[7] Blender, Open source 3D graphics creation.
http://www.blender.org/

[8] 3ds Max, Autodesk.
http://www.autodesk.com/3dsmax/

[9] VectorWorks, Nemetschek.
http://www.nemetschek.net/

[10] Liste de sites Internet offrants des modèles 3D commerciaux, Museum Quality.
http://www.huntfor.com/3d/links/commodels.htm

[11] Liste de sites Internet offrants des modèles 3D commerciaux, Ultimate 3D Links.
http://www.3dlinks.com/links.cfm?categoryid=9&subcategoryid=90

[12] Liste de sites Internet offrants des modèles 3D gratuits, Museum Quality.
http://www.huntfor.com/3d/links/freemodels.htm

[13] Liste de sites Internet offrants des modèles 3D gratuits, Ultimate 3D Links.
http://www.3dlinks.com/links.cfm?categoryid=9&subcategoryid=91

[14] Banff New Media Institute, The Banff Centre, Banff.
http://www.banffcentre.ca/bnmi/

[15] Windows, Microsoft Corporation.
http://www.microsoft.com/windows/

[16] Linux, Linux Online.
http://www.linux.org/

[17] Mac OS X, Apple Computer Inc.
http://www.apple.com/macosx/

[18] CrystalEyes, StereoGraphics.
http://www.stereographics.com/products/crystaleyes/body_crystaleyes.html

[19] Définition d'un anaglyphe, Wikipédia : L'encyclopédie libre.
http://fr.wikipedia.org/wiki/Anaglyphe

[20] Dispositifs de positionnement, Polhemus.
http://www.polhemus.com/

[21] Dispositifs de positionnement, InterSense, Inc.
http://www.isense.com/

[22] Pinch Glove, Fakespace Systems Inc.
http://www.fakespace.com/pinch.htm

[23] The Visualization ToolKit (VTK), Kitware Inc.
http://public.kitware.com/VTK/

[24] Qt, Trolltech.
http://www.trolltech.com/products/qt/

[25] VTK_QT, Carsten Kübler.
http://wwwipr.ira.uka.de/~kuebler/vtkqt/

[26] The Matrix Reloaded, The Internet Movie Database (IMDb).
http://www.imdb.com/title/tt0234215/combined

[27] SourceForge.net
http://sourceforge.net/

[28] Le site Internet d'**Avatar**, Christian Dompierre.
http://sourceforge.net/projects/avatar

[29] Camera Calibration Toolbox for Matlab, Jean-Yves Bouguet.
http://www.vision.caltech.edu/bouguetj/calib_doc/index.html

[30] ARToolKit, Human Interface Technology Laboratory, University of Washington.
http://www.hitl.washington.edu/artoolkit/

[31] Sportvision Inc.
http://www.sportvision.com/

[32] Open Dynamics Engine (ODE), Russell Smith.
http://www.ode.org/

[33] William R. Sherman, et Alan B. Craig, *Understanding Virtual Reality : Interface, Application, and Design*, San Francisco : Morgan Kaufmann Publishers, 2003.

[34] David A. Forsyth, et Jean Ponce, *Computer Vision : A Modern Approach*, New Jersey : Prentice Hall, 2002.

[35] Jun Park, Suya You, et Ulrich Neumann, *Natural Feature Tracking for Extendible Robust Augmented Realities*, First IEEE International Workshop on Augmented Reality (IWAR98), USA, 1998.

[36] Jun Park, Suya You, et Ulrich Neumann, *Extending Augmented Reality With Natural Feature Tracking*, SPIE on Telemanipulator and Telepresence Technologies, Vol.3524, Nov. 1998.

[37] Takashi Okuma, Takeshi Kurata, et Katsuhiko Sakaue, *A Natural Feature-Based 3D Object Tracking Method for Wearable Augmented Reality*, In Proc. The 8th IEEE International Workshop on Advanced Motion Control (AMC'04) in Kawasaki, Japan, 2004, p. 451-456.

[38] Youngkwan Cho, Jun Park, et Ulrich Neumann, *Fast Color Fiducial Detection and Dynamic Workspace Extension in Video See-through Self-tracking Augmented Reality*, Pacific Conference on Computer Graphics and Applications, 1997, p. 168-177.

[39] Ulrich Neumann, et Youngkwan Cho, *A Self-tracking Augmented Reality System*, In Proc. VRST 96, 1996, p. 109-115.

[40] Ulrich Neumann, Suya You, Youngkwan Cho, Jongweon Lee, et Jun Park, *Augmented Reality Tracking in Natural Environments*, International Symposium on Mixed Reality - Merging Real and Virtual Worlds, Ohmsha & Springer-Verlag, 1999, p. 101-130.

[41] Andrei State, Gentaro Hirota, David T. Chen, William F. Garrett, et Mark A. Livingston, *Superior Augmented Reality Registration by Integrating Landmark Tracking and Magnetic Tracking*, Computer Graphics, 1996, p. 429-438.

[42] Marc Pollefeys, *Self-calibration and Metric 3D Reconstruction From Uncalibrated Image Sequences*, Ph.D. Thesis, ESAT-PSI, K.U.Leuven, 1999.

[43] Paul Heckbert, *Projective Mappings for Images Warping*, in Fundamentals of Texture Mapping and Image Warping, Master's Thesis, University of California at Berkeley, Computer Science Division, EECS Department, Juin 1989, p. 17-21.

[44] Steven J. Leon, *Linear Algebra With Applications*, 5th Edition, New Jersey : Prentice Hall, 1998.

[45] S. K. Nayar, S. A. Nene, et H. Murase, *Real-Time 100 Object Recognition System*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18, no. 12, 1996, p. 1186–1198.

[46] P. Viola, et M. Jones, *Rapid Object Detection Using a Boosted Cascade of Simple Features*, in Conference on Computer Vision and Pattern Recognition, 2001, p. 511–518.

[47] P. Milgram, et F. Kishino, *A Taxonomy of Mixed Reality Visual Displays*, IEICE Trans. Information Systems, vol. E77-D, no. 12, 1994, p. 1321-1329.

[48] Azuma, R.T., *A Survey of Augmented Reality*, Presence, vol.6, No.4, 1997, p.355-385.

[49] Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, et Blair MacIntyre, *Recent Advances in Augmented Reality*, IEEE Computer Graphics and Applications, Vol. 21, No. 6, 2001, p. 34-47.

[50] C.G. Harris et M.J. Stephens, *A Combined Corner and Edge Detector*, in Fourth Alvey Vision Conference, Manchester, 1988.

[51] J. Shi et C. Tomasi. *Good Features to Track*, In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94), IEEE Computer Society, Seattle, Washington, Juin 1994, p. 593-600.

[52] Vincent Lepetit, Pascal Lagger, et Pascal Fua, *Randomized Trees for Real-Time Keypoint Recognition*, In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), IEEE Computer Society, San Diego, CA, Juin 2005, p.775-781.

[53] Gilles Simon, et Marie-Odile Berger, *Reconstructing While Registering : A Novel Approach for Markerless Augmented Reality*, In Proc. IEEE and ACM International Symposium on Mixed and Augmented Reality, 2002, p.285-294.

[54] Jun Rekimoto, *Matrix : A Realtime Object Identification and Registration Method for Augmented Reality*, APCHI'98, 1998.

[55] Hirokazu Kato, et Mark Billinghurst, *Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System*, In Proc. the 2nd IEEE and ACM International Workshop on Augmented Reality '99, 1999, p.85-94.

Annexe A

Cadre mathématique

L'estimation de la pose à partir d'images nécessite l'établissement d'un cadre mathématique formel basé sur un modèle théorique de caméra dans lequel seront traitées ces images. C'est ce qui sera établi dans cette annexe.

A.1 Les coordonnées image

Soit I une image numérique constituée de m lignes de n pixels chacune. Cette image peut être considérée comme une matrice $m \times n$. Soit maintenant $p = I_{v,u}$ le pixel sur la v^{ime} ligne et la u^{ime} colonne de I, on définit les *coordonnées image* de p comme le couple (u, v).

Celles-ci peuvent aussi être vues comme les coordonnées de p dans un système de coordonnées discret dont l'origine est située dans le coin supérieur gauche de I et où l'axe des x positif va de gauche à droite et l'axe des y positif va de haut en bas. Par abus de langage, un pixel est parfois défini par ses coordonnées image de la façon suivante : $p = (u, v)$.

A.2 Les coordonnées homogènes

Selon la notation standard utilisée en géométrie Euclidienne, un point p de l'espace Euclidien à n dimensions $\mathbb{R}^n$ est exprimé sous la forme d'un vecteur de n composantes

$p = [p_1, p_2, \ldots, p_n]^T$. Cette notation très répandue et parfaitement suffisante pour la géométrie Euclidienne n'est cependant pas la plus conviviale qui soit pour ce qui est de la géométrie projective utilisée entre autres dans l'étude des transformations de projection effectuées par les caméras. Une notation tout aussi cohérente, mais apportant beaucoup de simplicité à la géométrie projective est la notation dite *homogène* [44]. Cette notation est utilisée pour exprimer les points de l'espace projectif à n dimensions dénoté $\mathcal{P}^n$. Avec cette notation, un point p de $\mathcal{P}^n$ est exprimé sous la forme d'un vecteur de $n+1$ composantes $p = [p_1, p_2, \ldots, p_{n+1}]^T$ dont au moins un des p_i est non nul.

En géométrie projective, le point $p = [p_1, p_2, \ldots, p_n]^T$ de $\mathbb{R}^n$ peut en fait être dénoté dans $\mathcal{P}^n$ par n'importe quel vecteur homogène $p_\lambda = [\lambda p_1, \lambda p_2, \ldots, \lambda p_n, \lambda]^T$, où $\lambda \neq 0$. Tous ces vecteurs p_λ de $\mathcal{P}^n$ forment en fait la classe d'équivalence de la représentation homogène du point $p = [p_1, p_2, \ldots, p_n]^T$ de $\mathbb{R}^n$.

Par construction, les coordonnées Euclidienne standards sont obtenues des coordonnées homogènes par une projection sur le plan $\lambda = 1$, ce qui s'effectue par une simple division :

$$\mathcal{P}^n \rightarrow \mathbb{R}^n$$
$$[x_1, x_2, \ldots, x_n, \lambda]^T \mapsto \left[\frac{x_1}{\lambda}, \frac{x_2}{\lambda}, \ldots, \frac{x_n}{\lambda}\right]^T$$

Afin d'éliminer cette étape inutile, il est normalement convenu d'utiliser $\lambda = 1$ lors du passage de l'espace Euclidien à l'espace projectif. Il est cependant important de souligner qu'il s'agit là d'un choix purement pratique puisque selon la notation homogène, les points $x = [x_1, x_2, \ldots, x_{n+1}]^T$ et $y = [y_1, y_2, \ldots, y_{n+1}]^T$ de $\mathcal{P}^n$ sont égaux si et seulement si il existe un scalaire non nul λ tel que $x_i = \lambda y_i$ pour $i = 1, 2, \ldots, n+1$. Ainsi, dans cette notation, $[x_1, x_2, \ldots, x_n, 1]^T = [\lambda x_1, \lambda x_2, \ldots, \lambda x_n, \lambda]^T$ pour tout $\lambda \neq 0$.

A.3 Les homographies

Tel que mentionné, la notation homogène est très utile pour l'étude des transformations projectives. Elle permet entre autres de facilement décrire une application linéaire très commune[1] en vision par ordinateur appelée *homographie*. Une homographie est une application linéaire d'un espace projectif vers un espace projectif, inversible et préservant la colinéarité. Les homographies constituent en fait les bijections préservant la colinéarité les plus générales qui soient.

[1] Cette notion est par exemple très bien couverte dans les travaux de Pollefeys [42] ainsi que dans ceux de Heckbert [43], deux ouvrages dont cette section s'est d'ailleurs grandement inspirée.

Soit H une homographie de $\mathcal{P}^m$ vers $\mathcal{P}^n$:

$$\begin{aligned} H : \quad & \mathcal{P}^m \rightarrow \mathcal{P}^n \\ & p \mapsto q = Hp. \end{aligned}$$

H transforme linéairement les points de $\mathcal{P}^m$ et peut donc être représentée sous forme matricielle. Avec la notation homogène, cette homographie H de $\mathcal{P}^m$ vers $\mathcal{P}^n$ est représentée par une matrice $(n+1) \times (m+1)$.

De plus, puisque les points $q = Hp$ et $\lambda q = \lambda Hp$ de $\mathcal{P}^n$ représentent en fait le même point de $\mathbb{R}^n$ (voir la section A.2 sur la notation homogène), les homographies sont toujours définies à un facteur d'échelle près. Les matrices H et λH représentent donc la même homographie pour tout scalaire non nul λ.

C'est d'ailleurs cette invariance à l'échelle qui assure que les homographies sont toujours inversibles. En effet, on a[2]

$$H^{-1} = \frac{1}{det(H)} adj(H).$$

Or $\frac{1}{det(H)} adj(H)$ et $adj(H)$ représentent en fait la même transformation à un facteur d'échelle près. Ainsi, afin d'inverser une homographie H, il est possible d'utiliser $adj(H)$, laquelle existe toujours, même dans le cas où la matrice H est singulière. Soulignons de plus que l'inverse d'une homographie est aussi une homographie.

Les homographies sont entre autres employées pour décrire la transformation permettant de passer d'un quadrilatère à un autre (voir figure A.1). Cette opération est très souvent nécessaire dans les solutions aux problèmes rencontrés en vision par ordinateur. Comme nous le voyons dans la section 3.4, cette opération est d'ailleurs utilisée dans une des étapes du processus d'estimation de la pose effectué par les algorithmes de *ARToolKit*.

Puisqu'il s'agit d'une transformation de l'espace projectif $\mathcal{P}^2$ vers lui-même, l'homographie permettant de transformer un quadrilatère en un autre est représentée par une matrice 3×3 :

[2] L'adjointe $adj(H)$ d'une matrice H est définie comme la transposée de la matrice des cofacteurs de H ([44]).

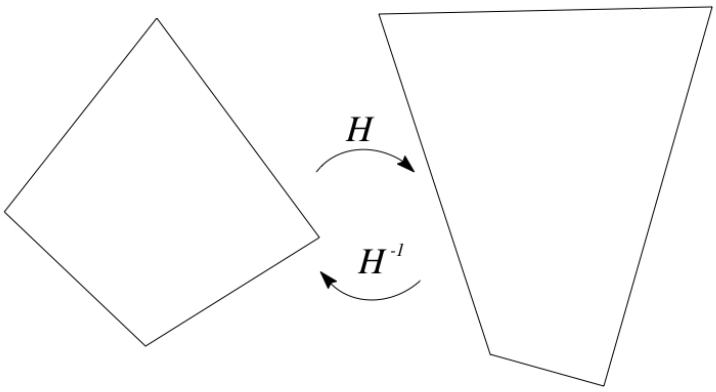


FIG. A.1 – Une homographie (et son inverse) permet entre autres de passer d'un quadrilatère à un autre très facilement.

$$
\begin{aligned}
H : \quad & \mathcal{P}^2 \to \mathcal{P}^2 \\
& p \mapsto q = Hp \\
& \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{w} \begin{bmatrix} uw \\ vw \\ w \end{bmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.
\end{aligned}
$$

Mais quelles sont les valeurs des éléments de cette matrice exactement ? Une matrice 3×3 a normalement 9 degrés de liberté. Cependant, comme les homographies sont définies à un facteur d'échelle près, la matrice recherchée n'en a que 8. Il est ainsi courant [43], [34] de fixer $i = 1$. Les valeurs des autres éléments de la matrice sont ensuite obtenues de la correspondance des quatre coins des deux quadrilatères en question.

L'analyse du résultat de l'application de l'homographie permet en fait de déterminer ces valeurs. Soit $p_i = (x_i, y_i)$ et $q_i = (u_i, v_i)$ où $i = 1, 2, 3, 4$ les coordonnées des 4 coins des deux quadrilatères respectivement. On cherche la transformation qui envoie p_i sur q_i pour $i = 1, 2, 3, 4$ (voir figure A.2).

Premièrement, les points p_i doivent être représentés en coordonnées homogènes :

$$[x_i, y_i]^T \mapsto [x_i, y_i, 1]^T.$$

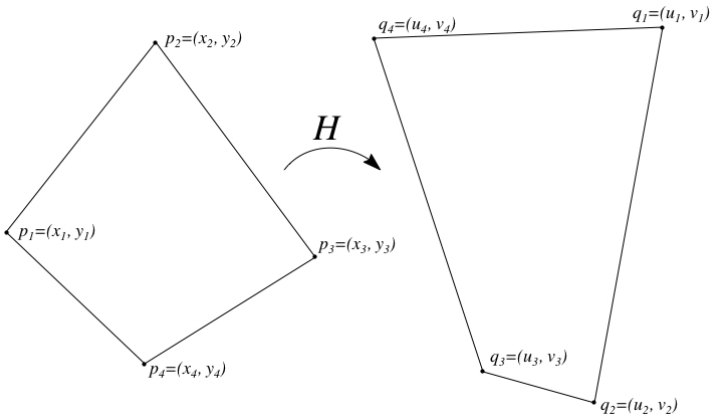


FIG. A.2 – L'homographie H recherchée envoie p_i sur q_i pour $i = 1, 2, 3, 4$.

L'homographie H est ensuite appliquée sur ces points de $\mathcal{P}^2$:

$$
\begin{aligned}
H : \quad & \mathcal{P}^2 \to \mathcal{P}^2 \\
& \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} u_i w_i \\ v_i w_i \\ w_i \end{bmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \\
& \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} u_i w_i \\ v_i w_i \\ w_i \end{bmatrix} = \begin{bmatrix} ax_i + by_i + c \\ dx_i + ey_i + f \\ gx_i + hy_i + 1 \end{bmatrix}
\end{aligned}
$$

Le résultat de la transformation doit ensuite être projeté sur le plan $w_i = 1$ afin de revenir en coordonnées Euclidiennes :

$$
\begin{bmatrix} u_i w_i \\ v_i w_i \\ w_i \end{bmatrix} \mapsto \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{ax_i + by_i + c}{gx_i + hy_i + 1} \\ \frac{dx_i + ey_i + f}{gx_i + hy_i + 1} \\ 1 \end{bmatrix}
$$

Toutes ces opérations produisent donc l'effet global suivant :

$$
[u_i, v_i]^T = \left[\frac{ax_i + by_i + c}{gx_i + hy_i + 1}, \frac{dx_i + ey_i + f}{gx_i + hy_i + 1} \right]^T
$$

Nous obtenons ainsi 2 équations par point :

$$
\begin{cases}
u_i &= \dfrac{ax_i + by_i + c}{gx_i + hy_i + 1} \\
v_i &= \dfrac{dx_i + ey_i + f}{gx_i + hy_i + 1}
\end{cases}
$$

c'est-à-dire

$$\begin{cases} gx_iu_i + hy_iu_i + u_i &= ax_i + by_i + c \\ gx_iv_i + hy_iv_i + v_i &= dx_i + ey_i + f \end{cases}$$

ou encore

$$\begin{cases} u_i &= ax_i + by_i + c - gx_iu_i - hy_iu_i \\ v_i &= dx_i + ey_i + f - gx_iv_i - hy_iv_i \end{cases}$$

Avec les 4 coins, nous obtenons 8 équations à 8 inconnues, lesquelles peuvent être réécrites sous forme matricielle par le système suivant :

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -y_2u_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3u_3 & -y_3u_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4u_4 & -y_4u_4 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2v_2 & -y_2v_2 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3v_3 & -y_3v_3 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4v_4 & -y_4v_4 \end{pmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}$$

Ce système d'équations linéaires peut être résolu par la méthode d'élimination de Gauss (par exemple) afin de nous donner les coefficients $a, b, c, \ldots, h$ composant la matrice H recherchée.

A.4 Le modèle mathématique de la caméra

La formation d'une image par une caméra est le résultat de la projection de l'espace 3D sur un plan appelé le *plan image* de cette caméra. Afin d'être en mesure d'étudier cette projection de façon mathématique, nous avons besoin d'un modèle mathématique de caméra. Afin de simplifier l'explication, considérons le cas du sténopé, un type particulier de caméra ne comportant aucune lentille et où tous les rayons de lumière passent par un même point appelé *centre de projection*. La figure A.3 illustre un tel appareil.

Considérons maintenant le système de coordonnées dont l'origine O est située au centre de projection et dont l'axe des Z est parallèle à la normale du plan image (voir figure A.4). Ce système de coordonnées est appelé le *repère de la caméra*. Tout objet du monde peut être exprimé dans ce repère et ce dernier peut en fait être considéré comme le repère du monde.

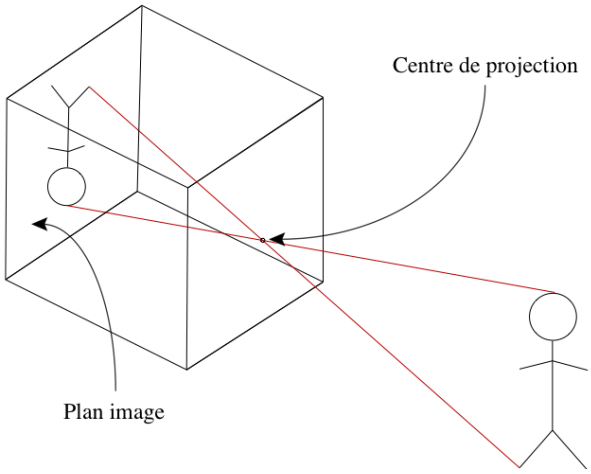


FIG. A.3 – Illustration d'un sténopé.

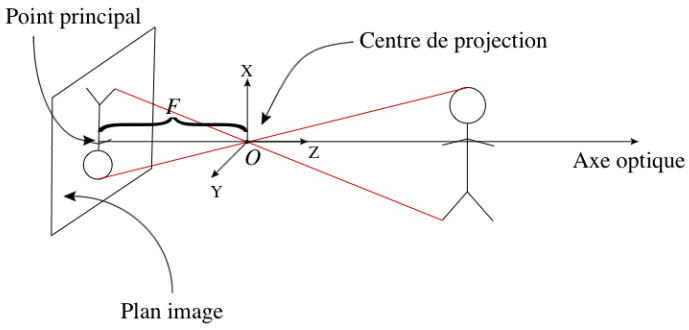


FIG. A.4 – Le repère de la caméra.

L'axe des Z est appelé l'*axe optique* et le point à l'intersection de l'axe optique et du plan image est appelé le *point principal*. Soit F la distance à laquelle est situé le plan image du centre de projection, le point principal a alors les coordonnées $(0, 0, -F)$ dans le repère de la caméra. Les explications qui suivent utilisent toutes ce modèle mathématique de caméra.

A.5 La projection de perspective

Considérons maintenant un point $P = (x, y, z)$ dans la scène observée par la caméra et son image $P' = (x', y'z')$ sur le plan image (voir figure A.5).

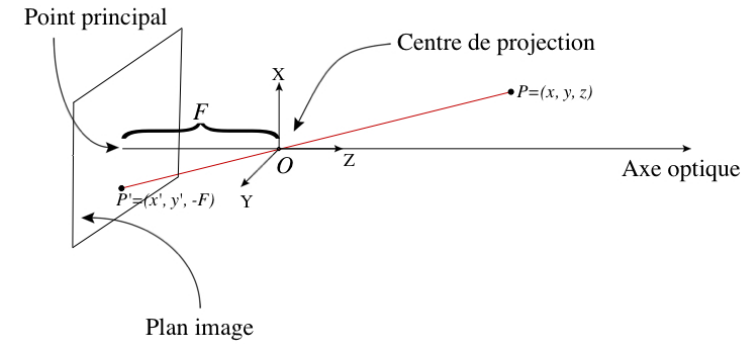


FIG. A.5 – Projection d'un point analysée dans le repère de la caméra.

Puisque ce point P' est sur le plan image, on a nécessairement

$$z' = -F$$

De plus comme les points P, O et P' sont colinéaires, nous avons

$$\overrightarrow{OP'} = \lambda \overrightarrow{OP}$$

c'est-à-dire

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Ainsi on a nécessairement

$$\lambda = \frac{z'}{z} = -\frac{F}{z}$$

Donc

$$\begin{cases} x' &= -F\frac{x}{z} \\ y' &= -F\frac{y}{z} \end{cases} \tag{A.1}$$

Ces deux équations nous permettent ainsi de déterminer les coordonnées de la projection (sur le plan image) de n'importe quel point exprimé dans le repère de la caméra. Cependant, ces coordonnées sont celles d'un système de coordonnées dont l'origine est située au point principal et dont les unités sont celles du repère de la caméra (qui peuvent être des mètres par exemple). Or, pour traiter des images numériques, il est plus courrant d'utiliser le système de coordonnées image (voir section A.1).

Afin de passer des unités du repère de la caméra (le mètre par exemple) à celle du système de coordonnées image (le pixel), il suffit d'introduire un facteur d'échelle dans

les équations de la projection de perspective (A.1). Cependant, puisque les pixels d'une caméra ne sont généralement pas carrés, mais plutôt rectangulaires, le facteur d'échelle en X n'est pas le même que celui en Y. Nous obtenons alors

$$\begin{cases} x' &= -aF\frac{x}{z} &= \alpha\frac{x}{z} \\ y' &= -bF\frac{y}{z} &= \beta\frac{y}{z} \end{cases} \tag{A.2}$$

D'autre part, le fait que l'origine du système de coordonnées image soit située dans le coin supérieur gauche de celle-ci plutôt qu'au point principal introduit deux paramètres de translation dans l'équation (A.2), à savoir u_0 et v_0, les valeurs correspondant aux coordonnées image (en pixel) du point principal. Les équations de la projection de perspective deviennent ainsi :

$$\begin{cases} u &= x' + u_0 &= \alpha\frac{x}{z} + u_0 \\ v &= y' + v_0 &= \beta\frac{y}{z} + v_0 \end{cases} \tag{A.3}$$

Finalement, pour prendre en considération les erreurs de fabrication qui font en sorte que l'angle θ entre les deux axes du système de coordonnées image n'est habituellement pas exactement de 90°, il est possible de démontrer [34] que les équations de la projection de perspective sont plutôt

$$\begin{cases} u &= \alpha\frac{x}{z} - \alpha\cot\theta\frac{y}{z} + u_0 \\ v &= \frac{\beta}{\sin\theta}\frac{y}{z} + v_0 \end{cases} \tag{A.4}$$

Il est facile de vérifier qu'avec la notation homogène, ces équations se traduisent en langage matriciel par

$$M = \begin{pmatrix} \alpha & -\alpha\cot\theta & u_0 \\ 0 & \frac{\beta}{\sin\theta} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{A.5}$$

En effet, nous avons

$$\begin{pmatrix} uw \\ vw \\ w \end{pmatrix} = M \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} \alpha x - \alpha\cot\theta y + u_0 z \\ \frac{\beta}{\sin\theta} y + v_0 z \\ z \end{bmatrix}$$

Ainsi, en divisant par la composante homogène pour revenir à la notation standard, nous avons bien

$$\begin{cases} u &= \alpha\frac{x}{z} - \alpha\cot\theta\frac{y}{z} + u_0 \\ v &= \frac{\beta}{\sin\theta}\frac{y}{z} + v_0 \end{cases}$$

La matrice M de l'équation (A.5) permet donc de calculer les coordonnées image de tout point $P = (x, y, z)$ du repère de la caméra.

Les paramètres α, β, θ, u_0 et v_0 sont propres à chaque caméra. C'est ce qu'on appelle les paramètres intrinsèques de la caméra. Il est possible de déterminer ces paramètres par une procédure appelée le *calibrage de la caméra*. Nous n'entrerons pas dans les détails du calibrage de caméra ici[3]. Notons cependant que, comme nous le voyons dans la section 3.4, *ARToolKit* utilise ces paramètres dans ses algorithmes d'estimation de la pose. Il est donc essentiel de calibrer la caméra utilisée avec *ARToolKit*. À cet effet, un programme de calibrage est fourni avec *ARToolKit*.

[3]Le lecteur intéressé par ce sujet est invité à consulter le travail effectué par Jean-Yves Bouguet [29]