Two-Handed Haptic Feedback in Generic Virtual Environments

THÈSE N^{O} 4342 (2009)

PRÉSENTÉE LE 19 MARS 2009

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE DE RÉALITÉ VIRTUELLE

PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Renaud OTT

acceptée sur proposition du jury:

Prof. P. Ienne, président du jury
Prof. D. Thalmann, directeur de thèse
Dr S. Coquillart, rapporteur
Prof. A. Frisoli, rapporteur
Prof. A. Ijspeert, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2009

"If I have eight hours to chop down a tree,
I'd spend six sharpening my axe."

Abraham Lincoln

Abstract

SINCE we hear about Virtual Reality as a discipline that could potentially provide benefits to many applications. Basically, the principle of Virtual Reality consists in stimulating user senses in order to give the impression to be in another place which they can discover and interact with. Today, most virtual reality systems create realistic visual and auditive environments. However the interaction with these environments can not be considered as natural. Indeed, we often use devices such as mouse, keyboard or joystick to move or manipulate them. These paradigms of interaction are in fact metaphors, which are not similar to reality. In some situations they are practical and efficient. However, the lack of intuitiveness sometimes makes them limited or simply ineffective.

To overcome this, researchers can use Haptic Devices. They are designed to simulate what is commonly called the "sense of touch", which includes more specifically, tactile, pain sense, thermal sense, and proprioception. Proprioception is knowledge gained by the perception of the relative member's position of the human body. In this thesis, we particularly focus on the simulation of proprioception. There are two advantages of such haptic devices. First, they can give the user more information on the nature of virtual objects (size, weight, finish, rigidity, etc..). Second, they can provide interaction paradigms that are closer to reality (three-dimensional interaction in a three-dimensional world). However, haptic device mechanics is complex. Moreover, proprioception is a sense that covers the entire body which is a rather large surface. For this reason, haptic devices usually apply force feedback on a very small portion of the body, such as fingertip. In addition to this hardware constraint, haptic research also faces software constraints. Indeed, a haptic application requires many computer resources in order to perform collision detection, dynamic animation of objects, and force feedback computation. Moreover, this should be done at a refresh rate that is much higher than the visualization for producing a convincing result.

In the first part of this thesis, we propose to increase realism and complexity of haptic applications. To achieve this goal, we use a state-of-the-art commercial device which allows to acquire the posture and position of both hands, and to apply forces on the fingertips and wrists. We propose techniques to calibrate and improve the comfort of these kinds of devices in order to integrate them into Virtual Environments. However, a two-handed haptic device do not presents only advantages. Indeed, It is much more complicated to compute forces on two hand models, than on a single point or fingertip. For this reason, in this thesis, we propose a framework to optimize this computation. Thanks to it, we can create Virtual Environments in which an object is graspable and dynamically animated by the laws of physics. When the object is seized by both hands, the haptic rendering engine realistically computes the forces on both exoskeletons. The efficiency of our rendering permits to apply these techniques to complex environments that have a significant number of objects. But the existing visual Virtual Environments are much more detailed than the ones seen in common haptic applications. In this thesis, we aim at reducing this gap. One of the problems is that these quality environments usually do not include specific haptic object properties, such as mass or material. We thus propose a software allowing even non-professional to quickly and easily add this information to an environment. Our results show that this haptic rendering engine does not suffer from the large quantity of objects. They demonstrate that we have an efficient framework for integrating a two-handed haptic interface into a generic virtual environment.

In the second part, we evaluate the potential of these kinds of Virtual Reality systems in more detail. While most applications can in theory take advantage of haptic devices, the practice shows that it is not always the case. Indeed, with experience, some metaphorical interaction paradigms remain more powerful than realistic ones. We thus present and study the integration of our two-handed haptic interface in a variety of applications. Evaluations show that depending on the application, it is not appropriate to reproduce reality: in teleoperation, for instance, simulating a virtual haptic steering wheel is less efficient than providing a force gesture interface. On the other hand, in virtual learning, the power of two-handed haptic manipulation is fully exploited and presents great advantages over standard techniques.

Keywords: Virtual Reality, Haptic Feedback, Computer Haptics, 3D Interaction, Virtual Grasping, Collision Detection, Dynamic Animation, 3D Modeling.

Résumé

DEPUIS de nombreuses années, on entend parler de la Réalité Virtuelle comme étant une discipline ouvrant de nombreuses perspectives pour notre société. La Réalité Virtuelle peut en effet potentiellement apporter des avantages à plusieurs types d'applications. Le principe de la Réalité Virtuelle consiste à soustraire l'homme du monde réel en lui proposant un autre monde - dit virtuel - dans lequel il peut agir. Le fait d'agir est très important car c'est ce qui rend le spectateur acteur. Etant donné que la perception du monde réel est réalisée par nos sens, il apparaît normal d'exciter directement ces mêmes sens pour proposer un monde virtuel. Ainsi, aujourd'hui, la plupart des systèmes de Réalité Virtuelle permettent de créer de manière relativement réaliste un environnent visuel et sonore, comme on peut le voir dans les jeux vidéos par exemple. Mais l'interaction avec ces environnements ne peut souvent pas être qualifiée de réaliste. En effet, on utilise souvent des périphériques comme la souris, le clavier ou les manettes de jeux pour se déplacer ou interagir. Ces paradigmes d'interaction sont en fait des métaphores, qui ne sont pas conforme à la réalité. Si dans certaines situations elles sont pratiques, elles peuvent au contraire aussi s'avérer peu efficace car non-intuitives ou tout simplement limitées.

Pour pallier cela, les chercheurs utilisent, quand ils le peuvent, des périphériques haptiques. Ces derniers ont pour but de simuler ce qui est communément appeler le sens du "toucher", qui englobe, plus précisément, la sensation du contact, de la douleur, de la température, et la proprioception qui est la connaissance acquise par la perception de la position des membres du corps humain. Dans cette thèse, nous nous intéressons plus particulièrement à la simulation de la proprioception. L'avantage des périphériques haptiques est double : d'une part ils permettent de donner à l'utilisateur un supplément d'information sur la nature des objets virtuels (taille, poids, état de surface, rigidité, etc.), et d'autre part ils permettent de proposer des paradigmes d'interaction qui sont plus proches de la réalité (interaction tridimensionnelle dans un monde tridimensionnel). Cependant, les périphériques haptiques sont généralement des machines dont la mécanique est complexe, et la proprioception s'applique le corps entier ce qui représente une énorme surface. C'est la raison pour laquelle un périphérique haptique donné ne couvre généralement qu'une toute petite partie du corps humain comme le bout du doigt. En plus d'une barrière matérielle, les chercheurs en Haptique doivent faire face à des considérations logicielles contraignantes. En effet, une application haptique nécessite de nombreuses ressources informatiques : il faut détecter des contacts, animer des objets virtuels, calculer des valeurs de forces, et ce, à une cadence bien supérieure à celle nécessaire pour les afficher.

Dans la première partie de cette thèse, nous proposons d'augmenter le réalisme et la complexité des applications haptiques. Pour ce faire, nous utilisons une machine qui est à la pointe de la technologie, en permettant de connaître la posture et la position des deux mains, et d'appliquer des forces au niveau des doigts et des poignets. Nous proposons des techniques pour calibrer et améliorer le confort de ce type de machine dans le but de l'intégrer dans des environnements virtuels manipulables. Ces caractéristiques matérielles exceptionnelles présentent cependant un inconvénient au niveau logiciel : il est bien plus complexe de calculer des forces sur deux mains que lorsqu'on utilise une machine ponctuelle. C'est la raison pour laquelle nous proposons dans cette thèse des algorithmes ainsi que des techniques pour optimiser les calculs. Grâce à ceux-ci, nous avons la possibilité de programmer des environnements virtuels dont les objets sont animés dynamiquement selon les lois de la physique. L'utilisateur peut ainsi les saisir, les manipuler et les lancer. Lorsqu'un même objet est saisi par les deux mains, le moteur de rendu calcule correctement les forces pour que les deux exosquelettes haptiques agissent conjointement. Dans un deuxième temps, compte-tenu de la vitesse de rendu due aux optimisations, nous proposons d'appliquer cette technique à des environnements complexes composés d'un nombre d'objets conséquents. Etant donné que la plupart des environnements de qualité n'incluent pas des informations spécifiques nécessaires au rendu haptique, telles que la masse ou le type de matériau des objets, nous proposons un logiciel d'édition permettant à quiconque de rajouter rapidement et facilement ce type d'information. Les tests montrent que notre moteur de rendu haptique ne souffre pas de la grande quantité d'objets. Ainsi, nous pouvons conclure que nous avons proposé un cadre de travail logiciel efficace pour intégrer des interfaces haptiques dans le but de rendre tangible des environnements virtuels qui ne sont pas spécifiques à l'Haptique.

Dans une seconde partie, nous nous intéressons de plus près à l'utilité et aux possibilités offertes par ce type de système de Réalité Virtuelle. Bien que la plupart des applications puisse en théorie tirer parti de ce genre de système, la pratique montre que ce n'est pas tout le temps le cas. En effet, certains paradigmes d'interaction métaphoriques se révèlent, après apprentissage, plus puissants que des paradigmes réalistes. Ainsi, la seconde partie de cette thèse se concentre sur quelques types d'applications, et présente des intégrations de système haptique à deux mains. Des évaluations montrent que suivant l'application ciblée, il ne convient pas de reproduire des objets réalistes, mais de penser de nouvelles interfaces. Ceci est peut-être le cas de la téléopération ou du prototypage. Tandis que d'autres applications telles que l'apprentissage virtuel, tirent vraiment parti de la puissance des périphériques haptiques a deux mains.

Mots-Clef : Réalité Virtuelle, Retour de Force, Rendu haptique, Interaction 3D, Manipulation Virtuelle, Détection de collisions, Animation virtuelle, Modélisation.

Remerciements

EN PREMIER lieu, je tiens à remercier chaleureusement le Professeur Daniel Thalmann. Avant même de le connaitre personnellement, il m'a transmis sa passion pour l'informatique graphique. C'est en découvrant ses cours en ligne il y a une dizaine d'années qu'il m'a incité sans le savoir à poursuivre des études dans ce domaine, puis à rechercher un stage dans son laboratoire. Il m'a ensuite proposé ce sujet de thèse si intéressant sur les interfaces haptiques, et donc donné par la même occasion l'opportunité de concrétiser un rêve. Je lui en serais toujours reconnaissant.

Je remercie aussi sincèrement le Docteur Frédéric Vexo, qui a eu très tôt confiance en moi, et qui m'a donné le coup de pouce et la volonté nécessaire au démarrage de cette thèse. Celle-ci n'aurait probablement jamais vu le jour s'il n'avait pas été là. Pour les mêmes raisons, je tiens à remercier le Docteur Mario Gutiérrez. Grâce à ces deux personnes, j'ai pu avoir l'opportunité de comprendre très vite ce qu'était le monde de la recherche. Ils ont su me transmettre les bases techniques pour écrire des articles et valider scientifiquement des expériences. D'autre part, ils ont été des amis avec qui j'ai eu le plaisir de partager les bons et les mauvais moments. Merci.

J'ai une pensée pour les reviewers de cette thèse, Docteur Sabine Coquillart, Professeur Antonio Frisoli, et Professeur Aucke Ijspeert, ainsi que pour le Président de mon jury, Professeur Paolo Ienne. J'ai beaucoup apprécié la pertinence de leurs questions, et leurs commentaires furent très utiles pour augmenter la qualité de cette thèse. Je remercie aussi Helena, Sylvain et Mathieu pour avoir corrigé mes horribles fautes d'anglais !

Ma reconnaissance va également à tous ceux qui ont travaillé avec moi ; ceux sans qui ce travail de thèse aurait été beaucoup plus long et difficile. Il y a d'abord Vincent De Perrot, que je n'arriverais jamais à féliciter ni à remercier suffisamment pour son excellent travail de master. Il m'a permis notamment d'effectuer une grande avancée pendant ma thèse en implémentant le modèle de la main et le Haptic Scene Creator. Je pense aussi à mon plus vieux copain de travail au laboratoire, Achille Peternier, qui, en me donnant accès à son excellent moteur graphique MVISIO, m'a permis de me concentrer uniquement sur le rendu haptique. Je remercie aussi Mehdi El Gaziani, pour avoir travaillé sur le Haptic Juggler (et sur bien d'autres choses), ainsi que Mireille Clavien pour son modèle 3D de la main. Enfin, j'ai une pensée pour Donald Knuth et Leslie Lamport, qui ont offert LaTeX au monde du logiciel libre, et aussi pour l'équipe de `Tigris.org` pour SVN.

Pendant ces quatre années de thèse, j'ai pu côtoyer deux générations de *VRLabiens*. Je remercie tout d'abord les "vieux" : Etienne, Benoit, Pablo, Bruno, Pascal, Sofiane, Patrick. Merci de m'avoir convaincu ! Ensuite je remercie bien entendu tous les “actuels” du labo : Sylvain, Xavier, Achille, Patrick, Mathieu, Damien, Helena, Jonathan, Barbara, Dani, Schubert, Ehsan, Anders, etc. Je pense aussi à Josiane Bottarelli et à Olivier Renault pour tout plein de choses qu'il serait difficile d'énumérer ici (la liste serait bien trop longue !), ainsi qu'à Ronan Boulic pour ses conseils avisés.

Je pense aussi à tous mes autres amis de l'EPFL avec qui j'ai partagé de nombreux repas, cafés, séances sportives et fou-rires : Pierre, Jean-Jou, Alex, Andrea, Gaël, François, mais aussi Roland, Vincent, l'autre Alex, Nikki, Mila, et j'en oublie certainement encore beaucoup (cette liste n'est bien entendu pas exhaustive. Si vous n'êtes pas dedans : je vous remercie aussi).

Je remercie aussi ma famille, notamment mon papa et Loyce qui, grâce aux bons restos de la région, m'ont bien engraissé chaque semaine ! Et merci surtout pour m'avoir donné l'opportunité de continuer des études : j'en ai bien profité !

Et je pense enfin à tous mes vieux “amis”, Nana, Ju, Raph, Raf, Antoine, Arno, Chup, etc. Je n'ai pas beaucoup été là pendant 4 ans, mais vous ne m'avez pas oublié. Merci !

Enfin, merci Ale d'exister... Je t'aime.

Je dédie cette thèse à ma grand-mère...

Contents

List of Figures

Part I

Preambule

Chapter 1

Introduction

As early as 1965, Ivan Sutherland, who is considered to be the father of computer graphics, defined the "Ultimate Display" [109]: *"It consists in a room within which the computer can control the existence of matter. A chair displayed in such room would be good enough to sit in."* This seminal paper introduced the key concept of complete sensory input and output using specific hardware capable of stimulating human senses in a similar way than in reality. However, most of the existing human-computer interactive systems have primarily focused on the graphical and auditory rendering of information, so much that some of them render the virtuality nearly as well as the reality. Nowadays, Ivan Sutherland is able to see the *virtual chair* he was dreaming about, but he will still experience some problems if he tries to sit on it.

1.1 Motivations

Haptic feedback is an attractive augmentation to visual display. The better virtual objects look, the greater is the desire of touching or manipulating them. These interactions enhance the level of understanding of complex data sets. Whenever we discover a new real object that does not seem dangerous, we instinctively want to hold it, touch it and feel it. The same instinct appears in Virtual Environments. However, the difficulty is considerable when dealing with the complexity of creating such stimuli. Therefore, no generic complete output tool exists that can be integrated easily into a virtual environment.

Two different obstacles can be distinguished in the creation of such a virtual haptic interactive system. The first one is to design the hardware device able to convey the desired sensory stimuli. To date, haptic peripherals usually propose to interact with a single fingertip. Even though it may seem to be a really small portion compared to the entire body, it nevertheless provides a much information, together with an efficient interaction. However these advantages have a cost. Indeed the second difficulty to overcome is to drive this hardware according to visual display and user state. Basically *touching* means going into contact with the objects and *manipulating* means moving, animating them. The collision detection and the dynamic animation both requires heavy computation. Moreover the time given for this computation is around the millisecond. Even for a fingertip, it is a complex problem to handle.

Today, haptic industry proposes devices allowing the user to interact with both hands. It appears to be really exciting for two reasons. First, when we look at the benefits of using a single fingertip to interact with a virtual environment, we can imagine that using the two hands represents a great improvement. Secondly, it give us a new challenge because as the interaction becomes more complex and takes more computation resources, it increases the difficulty to handle such devices.

This context motivates two research directions. First, we will investigate how to integrate a two-handed haptic device as an interface for improving the virtual environments, and then, we will perform a study on the advantages and drawbacks of such device into applications that commonly use haptic technologies.

1.2 Approach

As we have access to one of the rare two-handed haptic devices, it is easier to perform tests and validations. The two research directions have been performed together. We started from the low-level software device control in order to ensure a fast access, we then developed increasingly high-level tools together with applications in order to finish with a fully two-handed interactive system. Because such haptic devices are not really common, we faced problems that have never been raised before. However, we have been able to propose a solution each time.

1.3 Contributions

In this thesis, we propose various techniques to optimize and improve the simulation of touch by the means of a two-handed haptic device. We cover the entire software process needed to maximize interaction and the feeling of immersion.

This includes a study on the connection of a haptic device with the master computer running the simulation. Since force generation needs a fast refresh rate, we also present our method to optimize this computation by taking advantage of the new popular multi-CPU platforms. Then, we deal with the calibration of the device according to the focused applications. We also propose an interesting solution to improve user comfort when using the two-handed haptic device during long sessions.

At a higher level, we present our adopted solution to manipulate virtual objects with two hands. It is physics-based and is thus generic enough to allow a realistic bimanual interaction with many types of objects. The approaches that use scripts or heuristic analysis to simulate the grasping are efficient. However, they only handle specific cases and are thus not easily updatable. Moreover, the general approach that we use presents other advantages in term of force feedback computation or visual quality that are not negligible. However, the drawback of this physics-based approach is that it uses much more computational resources. Thus, we also propose optimizations in order to guarantee a constant refresh rate to achieve the desired results.

Moreover, this interesting property of being able to interact with various objects gives us the opportunity to exploit haptic feedback on a large-scale. We thus propose a study and the implementation of an authoring tool able to add haptic information to any kind of visual virtual environment. This need comes from the fact that designers as well as common 3D modeling software generally do not include haptic properties into their models. We show that adding this information into an existing environment is not trivial because a detailed mesh is barely suitable for haptic rendering algorithms. But we successfully address this issue.

This strong software rendering background allows to find out the potential of general two-handed Haptics in different kinds of applications. In the last part of this thesis, we present a teleoperation application, which is based on the remote driving of a mobile robot using the two-handed haptic interface. It provides interesting results stating that imitating the real user interface with such a device in this context is not necessarily the most efficient. We also present a mixed-reality training application, in which the user can manipulate virtual and real objects and assemble them. Finally, we describe an application allowing the user to visually and haptically discover a complex virtual environment.

1.4 Document Structure

This thesis is organized as follows. Chapter 2 presents an exhaustive list of haptic hardware devices. The main purpose is in fact to define common hardware properties that are useful to the understanding of the software. It then presents the main applications in Virtual Reality that take advantage of haptic displays. It also proposes an overview of the existing haptic frameworks allowing developers to quickly build haptic-enabled applications. Finally, we examine the state of the art in the two fields of handed interaction and two-handed Haptics.

The chapter 3 describes the two-handed haptic device that we use in this thesis. This device is a commercial product. We thus simply analyze each hardware component to understand how to take the maximum of their capabilities. This chapter also establishes the software needs that are explained hereafter.

Then, in the second part of this thesis, we deal with the technical solution retained to achieve a truly bimanual interaction with a generic virtual environment. First, in chapter 4, we address a common problem when dealing with haptic or tracking devices: the registration into the Virtual Environments. We explain the chosen solutions according to related work and context. Finally, we propose in this chapter an elegant solution to a user comfort issue that is experienced during long sessions. Then, in chapter 5, we mainly focus on the software control of the hardware. We present our multithreaded framework that allows the fast connection and force computation. It also contains a description of the collision detection engine used, of the animation, and of the visualization threads. These last components allow us to establish the basis of the hand model that we designed for interacting with various generic objects. In chapter 6, we give indications on the parametrization of this hand model, and present the specifications and the implementation of a powerful authoring tool intended to add haptic information to an existing environment. The goal is to be able to quickly develop an application to interact with an existing complex visual environment.

In the third part of this thesis, we perform various studies in terms of two-handed haptic applications. We separate these studies into two groups. In the first one, described in chapter 7, we examine the applications that need and take advantage of a realistic interaction paradigm. We present the integration of a two-handed haptic device in a Mixed-Reality environments made for assembly training. In this context, we study the possibility for a user to be able to interact with real and virtual objects at the same time. Then, in chapter 8, we propose a study in the field of teleoperation using unrealistic interaction paradigms.

The last part of this thesis presents the synthesis of our two major contributions. Primarily, we review the ability provided by our system to realistically manipulate and interact with virtual environments using our hands. Secondly, we state our recommendations concerning the type of user interfaces or metaphors to implement depending on the kind of applications.

Chapter 2

Haptics and Virtual Reality

THE WORD *Haptics* has been introduced by psychophysicists in the beginning of the 20th century. It was addressing the touch perception and the manipulation. Then, in the 1950's, it has been used for the teleoperation of remote controlled robots that send back contact force to the operator [52]. Finally, came the idea to substitute the remote controlled robot by a simulated system, which made the link between *Haptics* and *Virtual Reality*.

In this chapter we present the existing work related to the simulation of touch and proprioception in Virtual Reality. At first, in section 2.1, we present a general overview of common hardware haptic devices with their characteristics. We expect that giving these examples brings to the reader the knowledge of the terms, concepts and characteristics related to *Haptics*. Then, in section 2.2, we present a brief review of some applications that usually take advantage of haptic feedback. In section 2.3, we present the main existing software framework allowing developers to build virtual reality applications enabling haptic feedback. And finally, we focus in section 2.4 on the state of the art of the specific problem of two-handed haptic feedback generation that we will deal with along this thesis.

2.1 Haptic Hardware

Among neurologists, there is no consensus concerning the numbers of senses that the human being has. Traditional five senses (sight, smell, touch, hearing and taste) do not clearly represent every human senses. Haptic is of course associated to the sense of touch, but the impression of touching includes several *modalities*, including tactition, temperature, proprioception, or even physiological pain (Nociception). In 2.1.1, we present some haptic devices that stimulates these modalities. We present then proprioceptive haptic devices according to the stimulated body part (2.1.2), and finally we define *workspace* (2.1.3), *underactuation* (2.1.4.1) or *impedance/admittance* (2.1.4.2). Then, as we are focusing in this thesis on the proprioception, we will not cover anymore the other modalities.

2.1.1 Haptic Modality

In this section, we present the two main modalities usually studied in *Haptics*. The receptors associated with these modalities are not the same. Tactile is perceived though mechanoreceptors, temperature with thermoreceptors, and proprioception is believed to to be composed of information from sensory neurons of the inner ear, and of stretch receptors of the muscles and of the ligaments [102]. This variety of receptors increases the complexity of designing multi-modalities haptic devices.

2.1.1.1 Tactile and Thermal Feedback

The *tactile* feedback is related with the perception of object's surface. Human beings are able to feel the difference between pressures on the skin, but some parts, like the hands or the lips, are significantly much more sensitive than others. The drawing on figure 2.1 presents an homonculus whose part's size are proportional to their tactile perception.

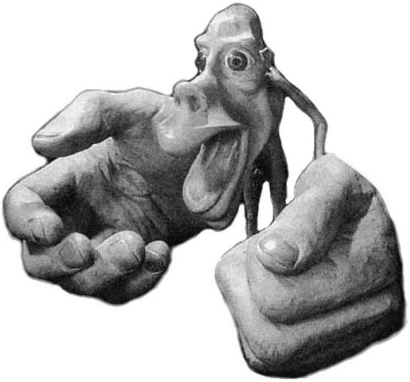


Figure 2.1: The Sensory Homunculus: this distorted human drawing reflects the relative space that body parts occupy on the somatosensory cortex.

Thus, most of the tactile displays focus on the fingertips. Several mechanical methods are used to stimulate them. Among them, the array of pins or of benders depicted in figure 2.2), is one of the most common, and appears to be the most efficient of his gender. Each of the pins moves independently at the normal of the human skin. To produce a real impression of feeling a specific object, the density and performance of the pins have to be adapted to the performance of the human tactile perception system. A common density is around one pin per square millimeter. Then, the size of the array varies usually from 3×3 pin to *hundredth* × *hundredth*. A detailed survey on tactile devices has been made by Benali *et al.* [12].

Thermoception is the sense by which an organism perceives external temperature. As for tactile perception, thermorececptors are also located in the skin. The computerized stimulation can be provided using radiation (IR and microwave), convection (air and liquid) or conduction (contact), or some combination of those [68, 13]. However, most of the thermal devices are based on conduction through Peltier cells [50, 38], and are only output devices: they do not send back the local temperature. Usually, the thermal devices are coupled with other mechanisms that have tactile or proprioceptive features [118], and it is rare to find exclusively thermal haptic devices in the literature.

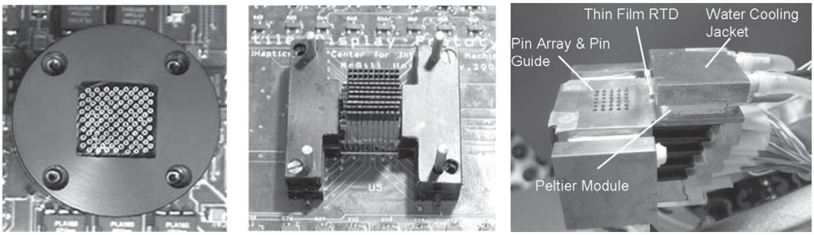


Figure 2.2: Three example of tactile Display. On the left, an array of pins *(McGill University)*. On the center, bending actuators *(McGill University)*. On the right, a tactile and thermal device *(Human-Robot Interaction Research Center, KAIST)*

In this thesis, we do not focus on thermal nor tactile feedback because they are usually implemented as output devices, and cannot be used as input devices for interaction purposes.

2.1.1.2 Proprioception

Proprioception comes from the Latin *proprius*, which means “one’s own” and perception. It is sometimes referred to the term *kinesthesia*, which is the sense of the relative position of the parts of the body. Unlike hearing or vision senses, which usually register environmental information, kinesthesia registers information about the body and the self. This is partly due to the fact that kinesthesia involves receptors located within the body, particularly in our articulations, muscles and tendons.

The haptic devices that are related with this kind of touch modality are described in the following sections as they are also categorized within the scope they can cover in the body parts.

2.1.2 Effect location on the body

Skin with its size of nearly $2m^2$ is a large organ. Moreover, it has an innervation up to hundreds of receptors per square centimeter. Thus, it is obviously difficult to design a device able to stimulate the entire body. A device like this would be too invasive and probably really expensive. This is why proprioceptive haptic devices focus on an small subset of the body.

2.1.2.1 Finger

Many haptic devices track and stimulate one or more fingers because they are the most controllable and sensitive body parts. Fingers also allow to perform difficult interactions in the real world; for example, grasping small objects, writing or feeling materials.

One of the haptic devices most widespread is the SensAble Phantom® family presented on figure 2.3. They provide from 3 to 6 DOF positional sensing and 3 DOF force feedback. The end effector could be a stylus or a thimble gimbal. These devices have a workspace varying from 16cm × 12cm × 7cm for the Phantom Omni®, to 80cm × 60cm × 40cm for the Premium 3.0.

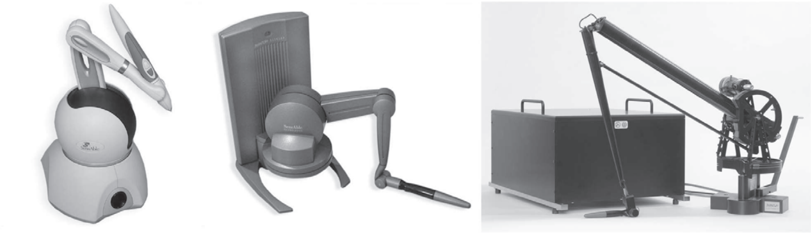


Figure 2.3: The Phantom® family. From left to right: Phantom Omni, Phantom Desktop and Phantom Premium.

Other devices with specifications comparable to the Phantom®, are produced by Force Dimension. Located in Switzerland, this company proposed two products in 2008: the Delta, and the Omega shown on figure 2.4. The Delta.6 has 6 DOF tracking and could also provide force and torque on 6 DOF. The maximal force is around 20 N.

Under this category, we can also cite the Cybergrasp$^{\text{TM}}$, which is a hand exoskeleton that provides 0.5 DOF force feedback on the five fingers. A more detailed description of this device is provided in section 3.2.2.

Finally, we have to mention a pure input device, which is extensively used in haptic interactions: the *Dataglove*. The goal of a *Dataglove* is to measure the hand posture in order to reproduce it in a virtual environment. In [107], Sturman *et al.* presents a detailed survey of this kind of glove, and their application.

2.1.2.2 Wrist

Among the devices that are intended to act in the wrist, we can cite the CyberForce force feedback exoskeleton. It uses the CyberGrasp$^{\text{TM}}$ support extension, and applies the force

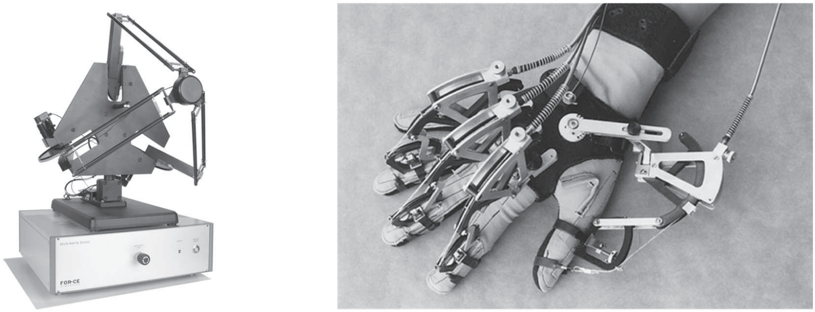


Figure 2.4: The Force Dimension Delta on the left and the Immersion CyberGrasp on the right.

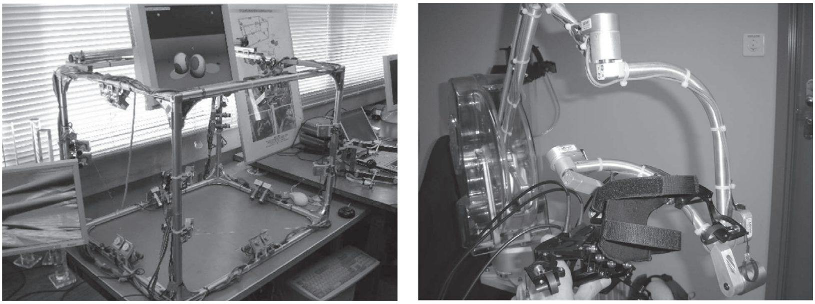


Figure 2.5: On the left, a SPIDAR system. On the right, a CyberForce combined with the CyberGrasp.

at wrist level. More details about this device are given in section 3.2.4.

Another interesting device is the Spidar system [62, 70] of the Tokyo Institute of Technology, shown on figure 2.5. Due to its unique simple design, proposes a very low inertia, and could be easily extended to bigger or smaller workspace as shown for instance in the Stringed Haptic Workbench [111].

2.1.2.3 Arm

Wrist haptic devices apply forces on the wrist, making thus possible to force the user to move his entire upper limb. However, we do not consider them as arm haptic device: an arm haptic devices includes the tracking of the position of the elbow and can even constrain the arm and the forearm into a specific posture.

Among arm haptic devices, we can cite two examples by PERCRO, in Italy, presented on figure 2.6. The first one, the Glad-in-Art exos, dates from 1993 and is a 5 DOF exoskeleton. The second one, the Pureform, could be considered as the evolution of the first one [47]. It has 5 DOF, including 4 actuated ones, and provides up to 100 N force, with a stiffness of 2×10^3N/m. Two end-effector configurations exist: with a handle, or with a two-fingers force feedback exoskeleton.

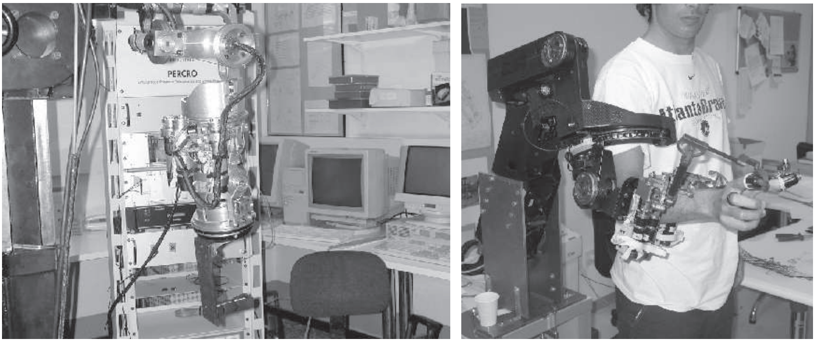


Figure 2.6: Two arm exoskeletons made at *PERCRO, Scuola Superiore Sant'Anna, Italia*. *(Reproduced with Permission)*

2.1.2.4 Rest of the body

Other parts of the body are also concerned by kinesthetic simulation. A nice example is shown on figure 2.7. The "Rutgers Ankle" haptic interface provides 3 DOF orientation sensing combined with a 3 DOF force feedback. This device, together with Virtual Reality-based exercises, presents interesting results in the rehabilitation of ankle's injuries [37].

Other example is the Food Simulator[66] of the university of Tsukuba, which is more well-known because it is one of the rare devices stimulating taste sense using chemical display. But, it also displays biting forces: a 1 DOF actuator generates a force on the user's teeth, and a sensor get the configuration of the mouth.

2.1.3 Workspace

Other characteristic to describe haptic devices is the workspace. The workspace is the space where the haptic device is able to track and generate force feedback. It should also take into account positions /postures that are impossible to reach/perform due to the mechanical configuration of the device. Usually the workspace is measured according to the body scope where force feedback is applied (finger, wrist, arm, etc), as described in the previous section.

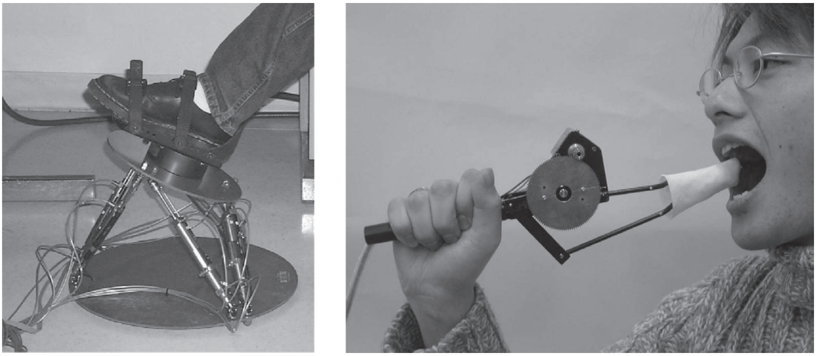


Figure 2.7: The “Rutgers Ankle” Rehabilitation Interface on the left, and the “Food Simulator” on the right. *(Reproduced with Permission)*

For example, the Pantograph, which should be more considered as a tactile device, has a workspace size equivalent to the maximal finger displacement when the wrist is constrained. Thus, in most cases, the user never reaches the limit and do not feel a constrain movement.

Other devices, such as the Phantom® Desktop^{TM}, requires the movement of the hand for interacting. Its workspace (16cm × 12cm × 12cm) is much more limited than the possible displacement of the hand using the arm. If such a quantity of movement is required, two solutions are conceivable. The first one consists in using greater size hardware, like the Phantom Premium, or to mount the Haptic device on another device that has a mobile base, as proposed by Lee *et al.* [75]. A second solution could be to use software control. This means when the user approaches mechanical limits of the device, the control modes switches to a displacement of camera instead of a displacement of interaction point [41].

Devices proposing a much bigger workspace, are for example, the human-scale Spidar systems, called Big-Spidar. This device allows user to move within a workspace of 3m × 3m × 3m [29]. The user is able to simulate a shoot when playing basket ball. However, he is unable to make a turn around himself because of the string, showing many limits in terms of rotation of the workspace.

2.1.4 Mechanics

As seen in the previous examples of haptic devices, many mechanical solutions exist when dealing with the conception of a haptic display. We can distinguish devices working with strings, or exoskeleton, while some others are magnetic [16]. In this section, we define two relevant concepts for this work, that are inherent mechanical properties of these devices: Underactuation and Impedance versus Admittance.

2.1.4.1 Underactuation

Nowadays, most of the haptic devices have less actuators than sensors. In this case, such a device is considered as an *underactuated haptic device*. The reason of this difference is that a sensor is smaller, weights less, costs less, and is easier to set up than an actuator. However, having more sensors will just let the user to have more freedom to explore the environment, but it does not improve the haptic feedback due to the lack od actuators. An underactuated device could suffer from several drawbacks due to the interactions are energetically non-conservative, as described by Barbagli *et al.* in [9]. To overcome these obstacles, Lécuyer *et al.* [73] proposed a technique for improving contact perception.

2.1.4.2 Impedance vs. Admittance

Another characteristic of a haptic device is to know if it is passive/impedance or an active/admittance device. Basically, an impedance device senses a position and command a force, whereas an admittance devices sense force and command a position [114]. Usually, admittance device are more effective to render stiff objects and high forces. An example of admittance device is the HapticMaster of FCS [36]. With this device, the user is able to hit a stiff virtual table without having the feeling of bounciness that usually happens with passive device. According to Van der Linde *et al.*, the stiffness is in the order of 10×10^3N/m. However, impedance control devices presents also advantages versus admittance devices, they are also usually lightweight, backlash free, able to render low mass [1], and cost less. We often say that admittance and impedance are dual in term of control.

The main problem when controlling passive haptic devices, is that the computation of the force/torque is not trivial when we want to move the user's body part attached to the end-effector. This is because this force/torque, which can be decomposed in components representing direction, orientation or magnitude, depends of the user himself. We have to extrapolate if *he is resisting ? Is he trying to move in the opposite sense? Or, is he already moving in the wanted direction?* etc. To provide a realistic force feedback, we should answer to these questions.

2.1.5 Other parameters of classification

Another interesting parameter of classification is the cost. We can find commercially available device starting at USD200.00$ (in the year 2008) like the Novint Falcon, and others that cost more than USD300,000.00$. It is a major consideration especially when we focus on a specific application.

Although it may seem difficult to measure the performances of the interfaces. Hayward *et al.* dressed a list of performance measure, which includes the number of degrees of freedom (including both sensed and actuated DOF), the motion range, the output forces and torques, the peak end-effector acceleration, the resolution of sensing, the refresh rate, the inertia and damping [60].

This section presented several relevant characteristics of haptic devices together with examples of existing devices. In the next section, we focus on the applications that uses them.

2.2 Virtual Reality Applications using Haptics

The use of haptic feedback on Virtual Reality applications can significantly improve the immersion in the Virtual Environments and make them more believable. There are different application scenarios where haptic feedback has an important impact. In this section we grouped these scenarios and we describe how haptic feedback is displayed.

2.2.1 Teleoperation

The first application is the *"Teleoperation"*. In fact, we could not strictly consider teleoperation as being a kind of Virtual Reality application. However, we recall that Haptics inherits from teleoperation in the sense that the first haptic interfaces were made for remotely controlling a distant robot or vehicle. Atkin *et al.* [2] define *telepresence* with this example: "*At the worksite, the manipulators have the dexterity to allow the operator to perform normal human functions. At the control station, the operator receives sufficient quantity and quality of sensory feedback to provide a feeling of actual presence at the worksite*". Today, haptic applications are still related to teleoperation if we consider that the *manipulators* given in this example are purely virtual.

2.2.2 Graphical User Interfaces

Another interesting field of research using haptic display concerns the improvement of the graphical user interfaces [80]. Having haptic feedback is a good way to alert the user that he is hovering a button or leaving a window. This makes much more sense if we consider a visually impaired user, or someone not used to manipulate a mouse. At usage, studies show improvements, in terms of speed for example [63].

In this field, we can also cite the work of Snibbe *et al.* [104], who presented a set of techniques for haptically manipulate digital media such as video, music or graphic. These interfaces focus on intuitiveness: the user knowledge of the reality could be directly applied to the interface.

2.2.3 Training

Virtual Environments are also extensively used in training. Training consists in increasing the ability of performing real tasks but in a context which is purely educational (i.e. not in the final context). There are many situations where it is difficult to perform training in real conditions, because it can be dangerous, expensive, or logistically difficult. Some Virtual training applications are improved with the use of haptic displays. We can cite the training of pilots: the Boeing flight simulator takes advantage of mobile platforms and haptic controlled sticks. Training surgeons is also difficult because of the obvious repercussions in case of mistakes on a real patient. Thus, medical simulation through Virtual Environments are frequent. Here we can cite specific haptic devices which aims at simulating laparoscopic surgery [11]. Immersion Corporation proposes many haptic devices for this purpose.

2.2.4 Games

Of course there are some haptic devices purely developed for entertainment industry. For example, force feedback joysticks and steering wheel greatly increase the immersion into computer games. More recently, the Falcon Novint haptic device, which is the cheapest 3 DOF display, is used with many computer games. It should provide a new kind of interactions and will probably give new ideas to computer games developers.

2.2.5 Other Applications

In the previous paragraphes, we have presented some groups of Virtual Reality applications that are improved by the use of haptic technologies. We could also cite the rehabilitation [56] [37], the 3D design process [45] [72] [32] or the virtual prototyping [99] [64] [27].

2.3 Haptic Software

Before the year 1993, date of issue of the Phantom®, almost every existing haptic devices were nearly unique. Then the creation of an adapted haptic software was made on a *per-device* basis. The advent of commercial haptic devices, like the Phantom, opened a new field of research: the creation of haptic software, or haptic APIs (Application Programming Interface). In the same way as a **computer graphics engine** provides a set of programming tools for stimulating sense of **vision** using a computer, the **haptic software** proposes software methods for the sense of **touch**.

In this context, we can cite the General Haptic Open Software Toolkit (known as GHOST). It is a C++ API, that eases the ask of developing touch-enabled application using a Phantom®. Basically, it allows to deal with simple, high-level virtual objects with physical properties like mass or friction. It is a purely haptic rendering engine. Hence, the developer has to take care himself of the graphical rendering. Later, this library was replaced with OpenHaptics$^{\text{TM}}$, but as GHOST, it supports only the Phantom®and is still limited to haptic rendering.

The Microsoft *DirectX* framework provides also a kind of Haptic API in *DirectInput*.

Its goal is to allow game developers to integrate specific haptic behaviors into a computer game. It is a high level rendering engine with functions like *“Shake”* or *“Apply force to the left”*, that are not dependant of the device. Then, if the user is playing with a vibrotactile or force feedback, it is the *DirectInput* engine that takes cares of the conversion between the high level haptic sensation into low-level actuator control. This approach is interesting because it really eases the work of the application programmer. However, it cannot guaranty the results and it is limited to low DOF devices like steering wheels or joysticks.

Haptic libraries supporting different kinds of hardware are not so common. We can cite three of them: ReachIn [96], Chai3D [33] and H3D. The first one is a complete framework that allows the creation of Virtual Environments that combines haptic, audio and visual feedback. The supported hardware devices are the Phantom, the Force Dimension Delta, the new Novint Falcon low-cost device, and some others. However, ReachIn has a cost and is not open source. Thus it could not be extended to other devices by their purchasers. The second one, Chai3D contains more or less the same functionalities than ReachIn, but has the advantage to be an open source project. Several haptic researchers are already working with it. However, it is more focused on single point interaction devices like the Phantom®. Finally the third one, H3D, presents the advantage to implement many haptic renderers, including OpenHaptics or Chai3D renderer, and an implementation of the algorithm of Ruspini presented in [98]. However, it is still difficult to extend it to whole-hand interaction.

Finally, Immersion® provides a library called the Virtual Hand Toolkit (VHT). It contains calibration tools, a connection utility, and an API. This software is much more focused on the whole-hand than other libraries, probably because of the CyberGlove®, which is on of their main products. It connects also with tracking devices and has some force feedback capabilities. However, the developers of this library focused on the integration with software like Dassault Systemes Catia, or AutoDesk MotionBuilder, and unfortunately, the C++ API does not provide the same level of functionalities.

2.4 Two-handed Haptic feedback

As seen in the previous sections, we can distinguish two different kinds of haptic devices and applications. On one hand, the tools we use in the real life have been physically reproduced “as is” for interacting with Virtual Environments. For example, we include in this category the force feedback steering wheel, the laparoscopic surgery simulation tool or the Rutgers Ankle rehabilitation device. We notice that these devices focus usually on a single kind of application. It is obvious that the laparoscopic display is barely suitable for other applications. On the other hand, devices, like the Phantom®, are not explicitly designed for a particular application. In the context of virtual manipulation, for example, these devices present many advantages. The major one is to increase the learning speed and the performance because 3D haptic interfaces offer the possibility to directly move and orient virtual objects in the 3D space that is usually proposed to virtual reality users, providing a gain in both intuitiveness and immersion.

However, it is possible to go one step further. First, common haptic interfaces are usually point-based. To integrate the human hand into the Virtual Environment, there is the necessity to use a glove-based input. Moreover, most of the human real interactions are accomplished with two hands. In this section, we present a state of the art focusing on these two points.

2.4.1 Whole-Hand Interaction Techniques

For manipulation tasks, whole-hand input devices present great advantages over the single-point interfaces. If implemented properly, it is obvious that the best way to interact with virtual objects is by using a virtual hand [58]. However, this method requires manipulation models that are more elaborate, and thus represents a more difficult challenge in term of integration [83]. One of the main difficulty comes from the animation and the detection of collisions between the virtual hand and the environment. Collision detection in 3D environments is a broad topic and haptic rendering is strongly linked with this field. *"touching"* is indeed more or less *"colliding with the hand"*. In terms of collision detection, finding if a point (the interaction point of a Phantom haptic display) collides or is into other objects is less difficult than detecting if a hand model collides virtual objects.

One of the first integration has been proposed by Iwata in 1990 [65]. The detection of contact is made using 16 control points on the hand model. The object is declared "grasped" when the thumb and another finger is touching the object. This simulation runs at 4 Hz! In [15], Bergamasco *et al.* proposed a solution with realtime constraints, and computing friction and force applied on the virtual object to determine if the object is captured or not. In these solutions, when an object is grasped, its local frame is linked to the coordinate system of the hand, implying that both move the same manner. In [22], Boulic *et al.* provides solutions for manipulating a grasped object into the hand, allowing for manoeuvering it with the fingers. But this technique do not provide force feedback. More recently, Borst and Indugula presented a physically-based approach of the grasping and manipulation. This technique has the advantage to prevent visual interpenetration, and to allow fast computation of the force feedback [21].

2.4.2 Two-handed Interface

In practice, to have only one hand able to manipulate objects shows how much the second hand is important. To understand this fact, we refer as Guiard's analysis of human skilled bimanual action [55]:

- First, Users can effortlessly move their hands relative to one another, but it requires a conscious effort to move a single hand relative to an abstract 3D space. This is the case when users have only one haptic device.
- Second, using both hands takes advantage of the user's existing skills. Most tasks we

> perform in our everyday lives involve using both hands in asymmetric roles, not one hand in isolation. This is even true for tasks like handwriting.

A key concept of Guiard's model is that the preferred and nonpreferred hands (depending of left/right-handed) act together, but not by the same manner. This asymmetric division of the tasks allows the hands to work with improved results , comparing to what either hand could achieve by itself. For example, Guiard reports that *"the writing speed of adults is reduced by some* 20% *when instructions prevent the nonpreferred hand from manipulating the page"*. One might also argue that using two hands to operate an interface only adds complexity and makes an interface harder. But there are many compound tasks that uses a single cognitive chunk [28].

Several haptic research include the combined use of 3D graphical and haptic systems for interacting with at least two single-point interfaces. Barbagli *et al.* present in [8] a multifinger (2 fingers) haptic interface made with 2 Phantom®. We can also cite the Spidar G&G which combines two Spidar at the same time [84]. We already cite Immersion™or Goble *et al.* [51], who designed a surgery training/planning system. All these systems have something in common: they do not allow to interact with 2 virtual hands.

2.5 Conclusion

In this chapter, we first presented an overview of existing haptic devices. It allowed us to define, by the example, words or concepts which will be used in this thesis. We presented then a review of the main Software Haptic framework, followed by the kind of application enhanced by the use of haptic interfaces. Finally, we focused on the state of the art in *whole-hand* and *two-handed* interaction.

It allows us to put one's finger on the fact that there are not a lot of research about truly two-handed interaction. This is probably due to the lack of two-handed haptic devices. In this thesis we will try to present first results in this field.

Chapter 3

Two-Handed Interactive System

AT THE END of the previous chapter, we presented research that shows the advantages of using hands to interact with our near surrounding. Indeed, a hand allows to move an object in order to examine it on different angles, to grasp an object for using it as a tool, or to assembly it with another one, or even to throw it. It is obvious that the role of hand is really important for human beings. Charles Darwin told that human beings could not have reach its place in the world without the use of his hands [35].

Many Virtual Reality applications tend to simulate a real life situation. But most of these applications do not simulate properly everything that can be achieved with a hand. In this thesis, we propose to study a better way to integrate the human hand in Virtual Reality systems. This could be achieved by using dedicated hardware, combined with efficient software control. In the first section, we present an evaluation of the hardware needs for achieving our goals. Then we present the chosen device. And finally, we give a first view of the software needed to efficiently control a two-handed device.

3.1 Definition of Needs

The role of the hands can be divided into two groups: **feeling** and **interacting**. In this thesis, we will focus mainly on interaction, i.e. on the action that occurs when one or two hands have an effect upon one or more objects (and vice-versa). To enable the action of the hands on an object, we need:

- A system for acquiring the posture of the hands. Indeed, the posture is really important for manipulation, because it allows specific grasping of particular objects.

- A system for tracking the position and orientation of the hands in the 3D space. The orientation of the wrist is important and could even be considered as part of hand posture. If we cannot orient the hand, some objects are impossible to grasp.

- A workspace allowing to reach any position of the space close to the chest.

To enable the second part of the assertion, i.e. the action of an object on the hands (which is also related to **feeling**), we need:

- A system for simulating proprioception on the fingers and on the palm. Indeed, the force feedback prevents user's fingers to penetrate into the objects. Moreover, it provides added information about the nature of a virtual object, and finally it eases the grasping by offering the feeling of contact which is difficult to obtain only with visual/audio feedback.

- For the same reasons, we need a system for applying force feedback on the hands.

We believe that these are the minimal hardware requirements for performing realistic two-handed interaction. In the next section, we present the hardware device that suits these needs.

3.2 Hardware Description

Our choice of device is greatly limited by the hardware needs that we presented in previous section. Among commercial devices, only one retains our attention: the Haptic Workstation$^{\text{TM}}$ provided by Immersion®. It is the only one that offers whole-hand tracking together with force feedback. In fact, this workstation is a combination of already existing haptic devices: the CyberGlove®, the CyberGrasp, and the CyberForce. In this section, we present each components individually, and then we present a view of the complete system.

3.2.1 CyberGloves®, ...

The CyberGlove® of the Haptic Workstation is the input device for acquiring finger postures. It is an instrumented glove that provides up to 22 joint-angle measurements, shown on figure 3.1, by transforming finger motions into real-time digital data.

It features three bend sensors per finger (even for the thumb), four abduction sensors, a palm-arch sensor, and sensors to measure flexion and abduction of the wrist. Each sensor is quite thin and flexible enough to be virtually undetectable by the user: they do not resist to movement.

The angles are measured by the sensors with a resolution of 0.5°. The sensor raw data are quite linear as each sensor is calibrated in the factory. Thus, the maximal nonlinearity is set at 0.6% over the full joint range. Finally, it is possible to gather the data at a refresh rate of nearly 100 records/second.

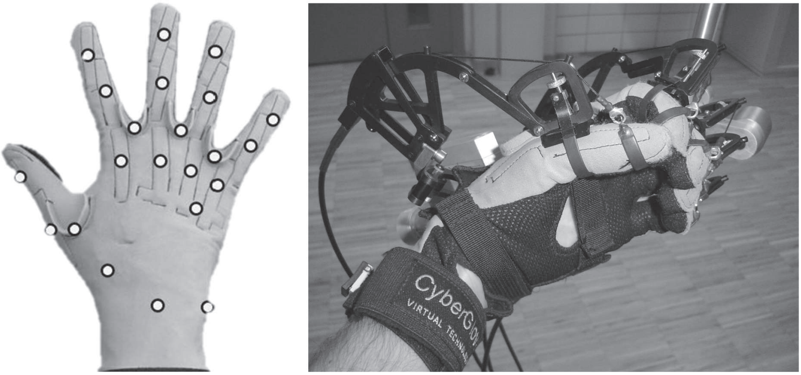


Figure 3.1: On the left: Sensor's location of the CyberGlove®. On the Right: The glove with the CyberGrasp$^{\text{TM}}$ exoskeleton

3.2.2 ... CyberGrasp, ...

The CyberGrasp is an output device for the fingers. It is based on an exoskeleton that fits over a CyberGlove and and adds resistive force feedback to each finger as presented on the figure 3.1. As its name suggests it, the objective of the device is to simulate the proprioception when a user grasps a virtual object.

Grasp forces are produced by a network of tendons routed to the fingertip via the exoskeleton. There are 5 unilateral actuators, one for each finger, and they can be individually programmed to prevent the user's fingers from penetrating into a virtual object. CyberGrasp exerts forces that are roughly perpendicular to the fingertips, whatever their position. The CyberGrasp cannot constrain the user to close his hands.

The CyberGrasp can produce a $12N$ force feedback on each finger, and the force can be set between $0N$ and $12N$ with a 12-bit resolution (4096 values). Finally, the weight of the exoskeleton is around $350g$.

3.2.3 ... CyberTrack, ...

The CyberTrack is an input device which provides 6D hand tracking. In fact, it is embedded into the CyberForce exoskeleton, and the name CyberTrack is only an abstraction of the device in the software. Thus, we can not consider that it is commercially available, as it comes with the CyberForce exoskeleton, the real name of the haptic device.

Anyway, this exoskeleton contains six optical encoders, allowing an accurate position and orientation tracking of a point located on the wrist. The position resolution is between

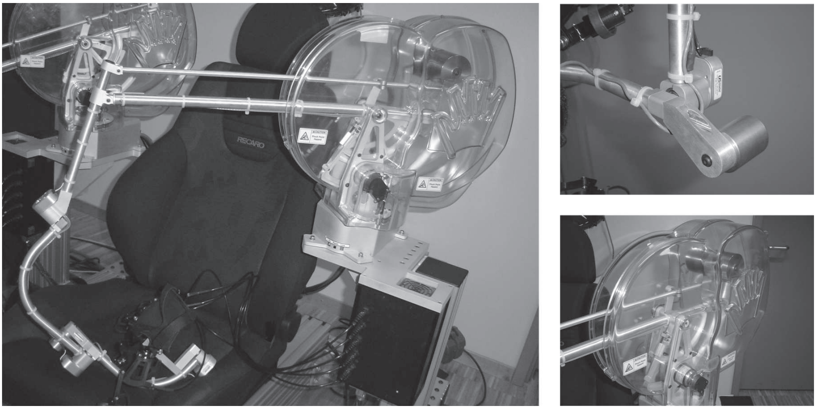


Figure 3.2: The CyberForce^{TM} exoskeleton

$60\mu m$ and $73\mu m$. The orientation resolution is around 0.09°. The refresh rate of these values is around $1kHz$.

3.2.4 ... and CyberForce ...

Finally, the CyberForce, presented on figure 3.2 is the output device which provides wrist 3D-force feedback. This allows the user to sense the weight and the inertia while picking up the virtual objects, or to feel the “relative” impenetrable resistance of a simulated table.

The device is an exoskeleton that can produce a maximal force of at least 7 N, depending on the armature configuration. The generated forces only affect the hand position, – i.e. it is not possible to oblige the user to rotate his hand–. Based on this consideration, we can say that the CyberForce is an underactuated device: it has less actuators (3) than degrees of freedom (6).

3.2.5 ... : The Haptic Workstation !!

The Haptic Workstation^{TM} is the combination of these four devices (two times for left and right arms), a seat and an armature to link everything together (see figure 3.3). This allows right and left whole-hand tracking and force feedback. For each upper limb, there are $22+3+3+5$ sensors and $5+3$ actuators.

A normally sized user (between 1,50 m and 1,90 m) could not reach the far limit of the workspace by outstretching the arms. However, it is impossible to bring the hands on the chest, and to fully extend the right/left arm on the right/left position. Moreover, it is usually

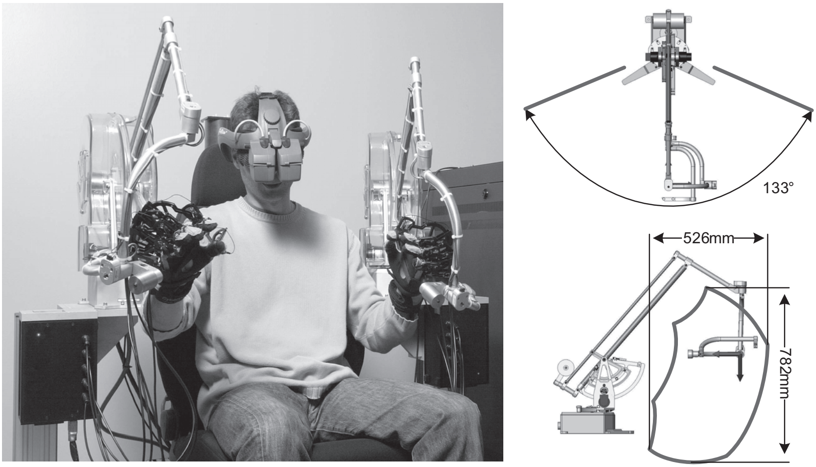


Figure 3.3: The Haptic Workstation^{TM} exoskeleton, and its workspace

tricky to cross the arms because of the exoskeletons. The schema on figure 3.3 presents the workspace of the Haptic Workstation^{TM}.

Finally, it is important to tell that the Haptic Workstation^{TM}, is controlled by two network connected computers (Linux kernel running on Pentium III 1GHz processor). Each computer is connected to a power amplifier which controls one CyberForce and one CyberGrasp exoskeletons, and to a CyberGlove via RS232. The figure 3.4 presents the schema of a "basic" VR system integrating the Haptic Workstation^{TM}. Usually, this configuration is the one used in this thesis. In the next section, we make a first presentation of the software required to create a two-handed interactive system. The software is running on the *Simulation Computer* of the figure 3.4.

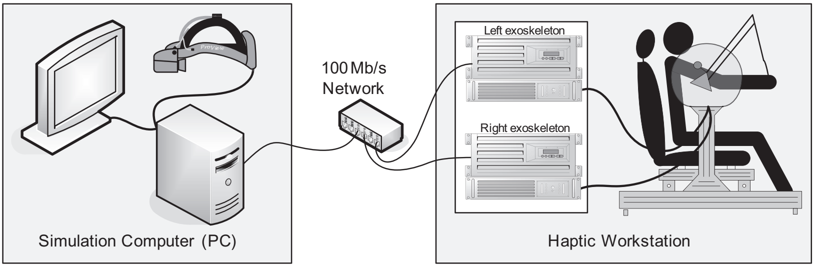


Figure 3.4: A common Virtual Reality System with the Haptic Workstation^{TM}

3.3 Software Control

In order to integrate a two-handed haptic device like the Haptic Workstation^{TM} in Virtual Reality applications, we need a kind of software driver. Even a mouse needs such piece of software. In our case, the driver could be considered as a piece of code giving access to the basic functionalities of the device by proposing functions/methods. Such functionalities could be, for example, to gather device encoder values or set a torque of one of the motors. However, this is by far not enough for creating a haptic-enabled Virtual Reality application.

In this thesis, we focus on providing an efficient software solution for using two-handed haptic devices. One of our goals is to integrate such device into applications in order to have a complete control over the virtual objects by using only the hands. To achieve this goal, we perform integration of new and existing techniques/libraries, and we choose to group them into a single framework. The idea behind, is that a student, a researcher or an application programmer could be able to get and to use every functionalities proposed in this thesis, by using only one software package.

In this section, we first analyze what could be exploited from a two-handed haptic device in Virtual Reality applications. Then we cite and present the components of the framework.

3.3.1 Haptic Software Requirements

The *"driver"* of a two-handed haptic device should be able to provide access to its lowest level. This includes the input values of the datagloves and the positions of the rotary encoders of the exoskeleton. It includes also the functions for applying force feedback. As a haptic device is usually connected via network, RS232, USB or Firewire with the master computer, the *"driver"* should embed the connection and the communication protocol.

Then, once these values are properly gathered, the device has to be calibrated according to the user and to the simulation. We can make the comparison with a computer mouse. We usually do not know the exact position of the mouse in the real world. But, the values returned by the mouse are relative to the previous state. If we want to know the exact position in a specific real coordinate system, we should find the transformation between mouse input values and the specified coordinate system. Same methods apply on haptic *input* devices. On the other side, the calibration of an *output* haptic device consists in proposing a way to create force vectors that are in the same space than input data, with consistent units.

Then, as a virtual reality application is usually working with visual feedback, the haptic software should offer the possibility to easily integrate the haptic device in the virtual environment. Thus a visual reproduction/model of the haptic end-effector – i.e. the hands – needs to be created and visually displayed. But displaying a virtual hand does not mean interacting. For interacting with virtual objects, we need to detect which of them are in contact with the virtual hand, and then we should animate these objects according to the contacts. Moreover, once collisions are detected, we need to compute force feedback and apply it on the output devices.

We believe that these requirements are mandatory for building a haptic two-handed interactive system. In the next section, we present the MHaptic framework which allows us to fulfill them.

3.3.2 The MHaptic framework

The Haptic Workstation^{TM} comes with the Virtual Hand Toolkit(VHT), which is a software framework to connect it, calibrate it, control it and even to integrate it into Virtual Environments. In theory, it meets the requirements defined in the previous section. However in practice, some components of this package are not suiting our needs. We found that the Haptic Workstation^{TM} could not provide its full potential when using this library. Moreover, the other haptic frameworks presented in section 2.3 are not properly addressing the whole-hand haptic interaction case. This convinced us to develop a new framework: MHaptic.

MHaptic is a framework composed of a library and an application. The name MHaptic has been chosen because it embeds and works together with the MVisio (Mental VISIOn) engine, a state of the art visual rendering system which contains many useful features for VR applications [95]. Moreover, this graphical engine also focus on the pedagogical aspect of 3D rendering, and thus is easy-to-use without compromising performances [94]. MHaptic is more or less made with the same state of mind than MVisio.

Here is a list of the components of the framework. We give more details on the implementation in the remainder of this thesis:

- A **connection** with the Haptic Workstation^{TM}. This may seem obvious, but the library has to establish a connection in order to gather input data (hand and finger positions) and to display force feedback. The connection and the refresh rate should be as fast as possible. We present it in 4.1 (page 31).
- A **calibration** module. It allows to quickly reconfigure the devices. It is particularly useful to adapt the Haptic Workstation to a new user without quitting the application. We deal with this module in section 4.2 (page 34).
- A **User Comfort Improvement** module. Being seated in the device could be exhausting after a while. This module increases the amount of utilization time before the fatigue. We present it in section 4.3 (page 41).
- A **Collision Detection** engine: this component determines if, where and when some objects come into contact during the simulation, either with the virtual hands or with other objects. As previously mentioned, our Haptic Workstation^{TM} stimulates the kinesthetic sensory channel. Thus, it is very important to detect all contacts in the VE in order to apply a fast and correct force-feedback to the user. We present it in section 5.2 (page 54)

- A **Dynamic Animation Engine**. This component is used for two purposes. The first one is to realistically animate the objects in order to have a believable manipulation tool. The second one is to ease the force-feedback computation and improve the graphical rendering of the hands. We will deal with this important feature in section 5.3 (page 58).
- A **Force-feedback Computation** engine: this ensures the correct computation of the forces. This part is a critical point since a wrong computation or an insufficient refresh rate leads to instabilities in the system resulting in lag or vibration. As stated in [49], such computation loop must run at least at $500Hz$ in order to be realistic. This constraint is respected in MHaptic.
- A **Scene Authoring Tool**: we remark that most of the existing 3D Scenes available contain only visual data. If we want to reuse these models, we need to provide a tool for adding haptic properties to the virtual objects. It is presented in section 6.2 (page 77).

In the first part of this thesis, we presented a review of the existing haptic devices. We also gave a review of the Virtual Reality applications that takes advantage of the force feedback and 3D interaction provided by kinesthetic devices. We also presented works related to whole-hand and bimanual interaction, from a cognitive and computer sciences point of view. It demonstrated a lack of truly two-handed haptic systems. Thus we studied the needs for naturally interacting with a Virtual Environment using both hands, and chose the Haptic Workstation^{TM} as hardware platform.

In the remainder of this thesis, we will present software solutions aimed at integrating such two-handed devices into Virtual Reality applications. For this reason, we proposed in this chapter a study of the software requirements and presented MHaptic, a framework made for this purpose.

Part II

Realistic Haptic Interaction

Chapter 4

Haptic Workstation$^{\text{TM}}$ Registration

MOST OF THE HARDWARE DEVICES require a driver or a piece of software for integrating them into computer applications. For example, a screen comes usually with its specific color profile. For a standard joystick, its neutral position and its end-points have to be calibrated. The haptic devices follow the same rule.

In this chapter, we present at first the way to get access to the data of a two-handed haptic device like the Haptic Workstation. Getting the input values (position, orientation or hand posture) and setting the output values (force feedback) has to be done with realtime constraints at a high refresh rate. We present the optimizations made for this purpose. Once we get or set these values, they need to be calibrated. This process in presented in the second section. Finally, in the third section, we present a solution for increasing the user's comfort while using the Haptic Workstation$^{\text{TM}}$.

4.1 Gathering Inputs and Affecting Outputs

Basically, a exoskeleton-based haptic device have internal registers which contain data representing the posture of the exoskeleton. And there are also registers containing the force or position values to be applied on the motors. Obviously, the first thing to do is to have access to them. This process has to be executed really quickly because it represents the upper bound of the haptic refresh rate. In this section, we present a study for optimizing the gathering of this data into computer memory. It concerns and focuses the Haptic Workstation$^{\text{TM}}$, but it could be applied on other devices.

As presented in 3.2.5, our haptic device is composed of two PCs (for left and right side) which are connected to the *Simulation Computer* through a dedicated 100Mb/s Ethernet network (see figure 3.4, page 25). A specific network protocol provided with the library Virtual Hand SDK is used to gather values or to apply force feedback. There are 4 functions, and we need to call them for each arm:

- for updating raw dataglove values (22 floats).

- for updating raw CyberTrack values (6 floats).
- for setting force values on the two device (8 floats).
- for updating string positions on CyberGrasps (5 floats).

Our goal is to maximize the update rate. Finding the maximal theoretical refresh rate can be achieved by evaluating the packets size and the speed of the link. Using a network protocol analyzer named *Wireshark*[1], we studied the behavior of the network. Results are presented on table 4.1. For each function we measured the size of the packet going from the computer to the Haptic Workstation$^{\text{TM}}$, the size of the replied packet, and the time taken to get the values after a call. We performed these tests 1000 times under the best conditions –i.e. without data treatment nor simulation, only a loop with function calls–. In the table, we present only the mean of the times, but we mention that the standard deviation is quite small (less than 5%), and thus, could be neglected.

Procedure	**Packet size**		**Time**
CyberTrack position update	U	86 bytes	507 μs
	D	135 bytes	
CyberGlove angle update	U	86 bytes	243 μs
	D	278 bytes	
Setting Forces	U	155 bytes	268 μs
	D	86 bytes	
CyberGrasp strings position update	U	86 bytes	248 μs
	D	114 bytes	
TOTAL	1026 bytes		1337 μs

Table 4.1: Ethernet link between the Haptic Workstation$^{\text{TM}}$ and the computer running the simulation.

Over a 100 Mbit/s link, if we consider that we have 1026 bytes (8208 bits) to transfer to get every data, we can have theoretically around 12000 updates per seconds. However, because these 4 calls takes 1337 μs, we are bounded to only 750 updates per seconds. Moreover, we remind that we are addressing only one side (left or right) of the device when dealing with these numbers.

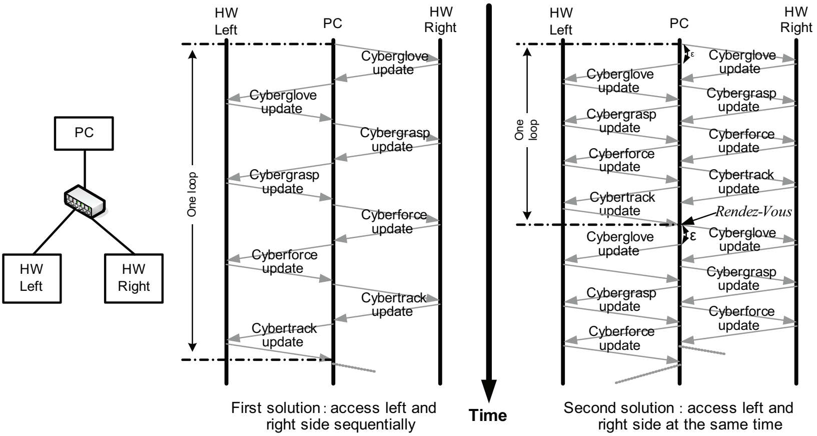
To perform the communication with the two sides of the Haptic Workstation$^{\text{TM}}$, there are two solutions. The *first* one is to sequentially access to the left side, to wait for the response, and then to do it again with the right side. So in this state, the two-handed haptic refresh rate falls to 375 Hz. This is too low and need to be optimized. The *second* one is to access simultaneously to the two sides. This is illustrated on figure 4.1. The first solution is the easiest to implement and guarantee that there will not be any packet collision, but the second solution should be faster... in theory. To implement it, we have indeed to face two problems: the call to one of the procedure for accessing to the data is a blocking call

[1] http://www.wireshark.org/ . In June 2006 the project was renamed from *Ethereal* due to trademark issues.

(*problem 1*); it means that the program waits for the response before continuing with the following code. Moreover, the packets can collide each other resulting in network flood that increases the latency (*problem 2*).the following paragraphs present the chosen solutions.

To solve the *problem 1*, we have only one solution. It consists in multitasking the accesses to the left and right device. Basically, we divide the Haptic Thread into two subthreads for left and right sides. To avoid synchronization issues resulting in uncontrollable different refresh rates for both sides, we establish a *rendez-vous* point at the end of a complete update loop.

This multitasking is the source of the *problem 2*: on a computer with several cores or processors, the time noted ε on figure 4.1 has great chances of being null, because the two subthreads could be running at the same time and have been waiting each other at the *rendez-vous* point. This implies that they are both trying to use the Ethernet controller at the same time. The operating system is able to manage that, but unfortunately not efficiently. Thus, we have to avoid the case. Although the solution seems to be trivial, – i.e. making that one of the two subthreads goes to sleep during a fraction of time ε – it appears to be almost impossible on the Microsoft Operating Systems. On Windows®, `Sleep(...)` procedure accept only millisecond argument and is moreover not precise (because it is not a truly realtime OS). To overcome this issue, we made two subthreads that are not completely equivalent: on the left arm we perform some computation concerning the force feedback before the first call to an update procedure, whereas the same computation is performed in the middle of the loop for the right arm. As this computation approximatively takes $50\mu s$, it is enough to shift the Ethernet port utilization. We remind also that this solution works because we have a Full-Duplex 100 Mbit/s Ethernet link.



First solution : access left and right side sequentially

Second solution : access left and right side at the same time

Figure 4.1: Network communication diagram.

Using these techniques, we achieved a mean refresh rate of 720 Hz, which is nearly optimal. We stated indeed earlier in this section that 750 updates per second is the theoretical upper bound. The loss of speed is probably due to the computations that may happen between the network calls.

But we also found that some of the input data (like gloves angle values, or grasp string positions) where not updated as fast as 750 Hz. This is confirmed by the factory description of the CyberGlove®, which states that a maximum of 149 records per second could be retrieved by RS232 with this device. Thus we do not need to get these data at each haptic loop. Consequently, we only update gloves values and string position every 6 loops. By doing so, we do not lose any meaningful data, and we increase the haptic refresh rate to **950 Hz**.

To put in a nutshell, we presented in this section some low-level optimizations for accessing to the Haptic Workstation^{TM} as fast as possible. We have shown that it is difficult to address faster this networked haptic device, because we are already near the theoretical optimal speed. We remind that the optimization concepts presented in this section could be extended to most of the haptic devices connected via a RS232, Parallel or Ethernet link.

4.2 Haptic Workstation^{TM} Calibration

Several applications of combined immersive visual and haptic displays greatly beneficiate from accurate calibration and registration. We can cite for example virtual prototyping, manipulation training or surgical simulation. Unfortunately, precise registration is difficult to achieve because of the large number of factors that affect overall system accuracy. In this section, we deal with the specific calibration of the Haptic Workstation's devices. We present the calibration methods used in the rest of the thesis to achieve an efficient two-handed haptic manipulation system.

The Haptic Workstation^{TM} is a combination of four devices. Each one need to be calibrated properly to be fully functional. As we will see it in the next subsection, many methods exist addressing many kinds of applications.

4.2.1 CyberGlove registration

The human hand is a remarkably complex mechanism and in the year 2008 the instrumented glove still appears to be the predilection device to map the real hand posture on a virtual model. There are different goals when dealing with the calibration of *datagloves*:

- To reach a perfect matching between many real and virtual hand postures, allowing for gesture recognition for example. This technique could be based on Hidden Markov Models, like the work of Lee et al. [76], or also on Neural Network algorithm like Xu [117] and even on genetic algorithms [108].

- To provide visual fidelity without aiming at determining exact values of real joint angles. This technique is often used because visual realism is a strong need in Virtual

Reality. Griffin et al. provides a method using a kinematic chain using least square regression iteration where the basic idea is to move the four fingertips against the thumb and to record the trajectory in order to perform the calibration [54].

Moreover to these different objectives, we can also distinguish different methods for performing the calibration in itself:

- To use only the instrument glove for the calibration and not external devices like trackers, cameras, etc. Indeed, most of the efficient techniques uses an external system to validate the calibration or to check the errors. Fischer et al. [44] used a stereo vision system to measure the real 3D positions of the fingertips, while also storing the joint sensor readings of the dataglove. Of course, these calibration protocols offers better performance but are much more difficult set up and thus are limited to particular applications.
- To minimize the time consumed to perform the complete calibration procedure. According to the literature, we can find methods which takes from few minutes [69] up to several hours [117].

As we are studying in this thesis two-handed haptic interactions, we will not focus on reconstructing a virtual hand with the best matching as possible. Indeed achieving visual fidelity is enough for our needs as we do not plan to recognize gestures or to reach extreme hand's postures for grasping objects. Moreover, we prefer to have a fast calibration process without needing external peripherals in order to increase the ratio *"Time of session"* ÷ *"Time to install the user"*.

The CyberGlove® is made of bend sensors. These sensors have a varying resistance in function of their bending. This resistance is not proportional to the angle, but Immersion® calibrates the glove sensors at their factory. Thus, the values returned by the black box connected to the glove should be proportional to the angles. It means that there is a linear function between glove angles and gathered values. We assume that this is true, and in practice, we have verified that the error is not significant.

A linear function has a form $y = mx + p$, it has only two parameters m and p. The goal of the calibration is to find them. We need at least two couples (x_1, y_1) and (x_2, y_2) to compute m and p. For a single sensor, the calibration procedure is easy to do. We place the sensor in two reproducible configurations that are sufficiently different, for example, *not bent* ($y_1 = 0°$), and bent at a *right angle* ($y_2 = 90°$), gathering at the same time the values of the black box (x_1 and x_2). Then is is trivial to find the m and p coefficient of this sensor.

However, in practice, we do not want to place each sensor in 2 different configuration. This would take too much time. We thus have isolated 4 different hand postures shown on figure 4.2. For each sensor, two of these postures are different enough to have a good estimation of m and p. For example, sensor A is calibrated with postures 1 and 3, whereas angle B uses postures 1 and 2. Finally, we mention that each time we register the values of a posture, we store them into the memory. By doing so, we can perform the calibration in the

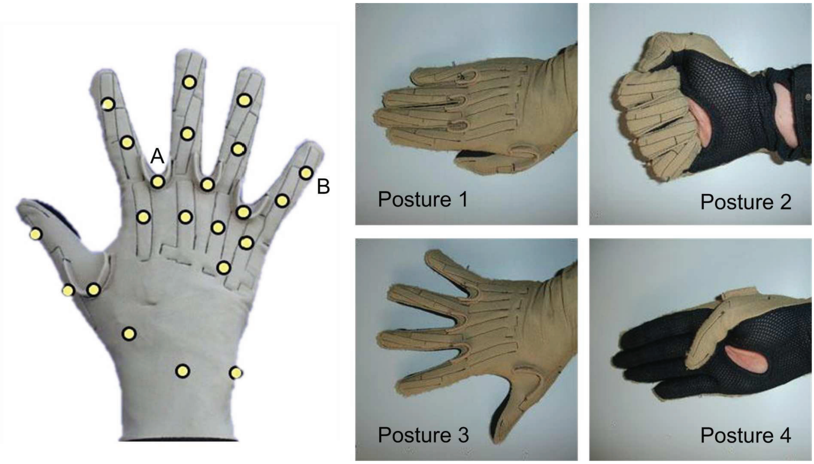


Figure 4.2: The four hand's postures used in the CyberGlove® calibration tool.

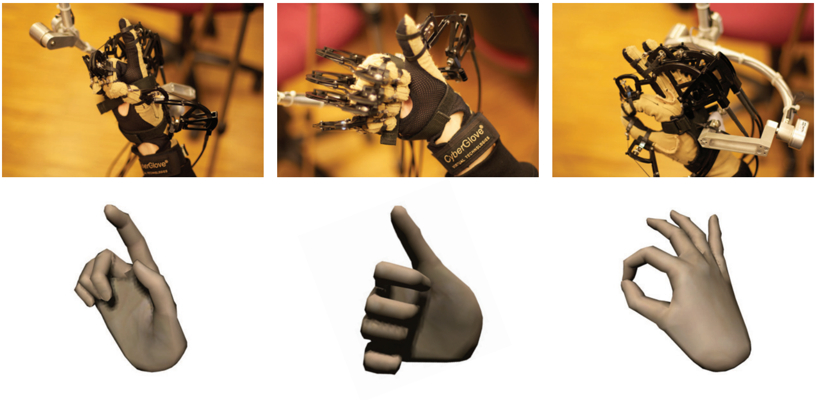


Figure 4.3: Comparison between real hand and virtual hand after calibration.

order that we want (and not necessarily 1,2,3 and 4), and we can make quick refinements of a single posture.

Despite the fact that this method is really simple, it is efficient. It takes less than a minute to fully calibrate the gloves with a user who never worn them. Visually, most of the postures are reproduced with a reasonable level of fidelity. We present on figure 4.3 comparisons between real and virtual hands after calibration using our method.

4.2.2 CyberTrack registration

The six optical trackers of the Haptic Workstation^{TM} are factory-calibrated. The Jacobian that transforms from this 6 degrees of freedom space into a cartesian space is already performed by the hardware. However, this cartesian space is relative to the configuration of the CyberForce exoskeleton at startup time. Basically, it means that the user has to start the Haptic Workstation in a predefined position in order to “calibrate” it. The problem is that it is impossible to reproduce perfectly this position at every startups, and moreover the trackers of the two exoskeletons are not in the same coordinate system.

The figure 4.4 summarize the situation after a normal startup.

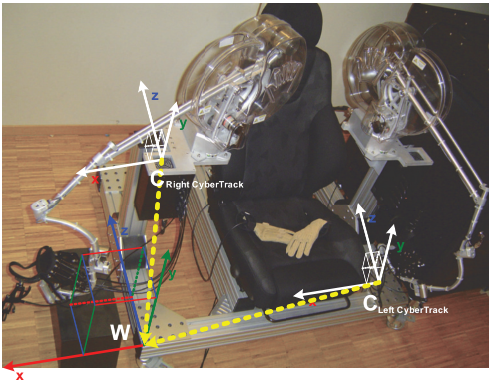


Figure 4.4: The exoskeletons reference frames when starting up the device

First, we can remark that the left and right coordinate systems $C_{\text{Right CyberTrack}}$ and $C_{\text{Left CyberTrack}}$ do not have their origin at the same position. Then, we can see that their axes are parallel together, and also parallel to the aluminium chassis of the Haptic Workstation^{TM}. Finally, we can mention that the unit of these coordinate systems is not according the *SI*. We are satisfied with the orientation of the coordinate systems with *y* for up/down, and *x* for left right, because from a user point-of-view, it reminds the OpenGL camera orientation. Moreover, it means that we do not need to convert the orientation value of the CyberTrack, but only the position value.

Thus, the registration procedure consists in putting the two exoskeletons in the same orthogonal coordinate system W with *SI* units, which has been arbitrary set at the position shown on figure 4.4. It means that, for each exoskeleton, the goal is to solve the unknowns in the matrix presented in equation 4.1. α represents the scaling and t the translation transforming raw data into calibrated positions in the coordinate system W. This matrix has 6 DOF, so we can deduce that at least 6 correspondences $P_{C_x} \longrightarrow P_W$ are necessary to find the 6 unknowns. As each position has already 3 dimensions $((x,y,z))$, we only need to find two configurations of the exoskeleton that are easily reproducible without error and whose position in the real world is known (have been measured by us).

$$P_{C_x} = \begin{bmatrix} \alpha_x & 0 & 0 & t_x \\ 0 & \alpha_y & 0 & t_y \\ 0 & 0 & \alpha_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times P_W, \text{where each } \alpha_x, \alpha_y, \alpha_z > 0 \tag{4.1}$$

In practice, we found that the scaling parameters α_x, α_y and α_z where close to $0.01 \pm 5\%$. This let us think that the hardware calibration returns positions with a unit length that is the centimeter. Moreover, we found that the value of α_x, α_y and α_z was not dependent of the startup procedure. Thus, we can hardcode these values and the goal of CyberForce registration is only to get t_x, t_y and t_z. This give us the opportunity to drop one of the two calibration configurations of the exoskeleton.

To validate our registration, we use a PhaseSpace Motion Capture system. This system is composed of optical cameras and LEDs. Up to 128 LEDs could be tracked in a predefined coordinate system up to a refresh rate of 480 Hz. We calibrate the system using 4 cameras to track a single LED attached to the CyberForce into the same coordinate system than the Haptic Workstation^{TM}. This setup is shown on the picture of the figure 4.5. Then, we measure pairs of positions gathered from the PhaseSpace and the calibrated CyberTrack. The goal of our calibration is of course to minimize the distance between each pair of positions. We try to move the hand into the complete workspace, and we also orient the hand in different positions. We aim to collect a pair each 10 ms during 5 minutes, but sometimes, occlusions prevent to optically capture the LED. Thus, we finally record “only” 23000 pairs. To interpret this great amount of information, we voxelize a virtual cube in the workspace, then we randomly pick a pair belonging to each voxel, and finally, we display the *“difference vector”* of this pair representing the difference between the CyberTrack and PhaseSpace positions. We perform this random picking several times. One of the result that we get is presented on figure 4.5.

The most noticeable conclusion is that the scaling parameters α_x, α_y and α_z that we previously set to 0.01 are quite correct. If they were not, two *difference vectors* that are close should be pointing the more or less same direction, and this effect should affect all *difference vectors*. Then, we visually did not remark any *pattern* affecting every *difference vectors*, which could came from a calibration error. Moreover, the mean length of these *difference vectors* is around 2 cm, with a standard deviation of 9 mm. We mention also that an optical tracking system is by nature much more error-prone than the trackers of the exoskeleton.

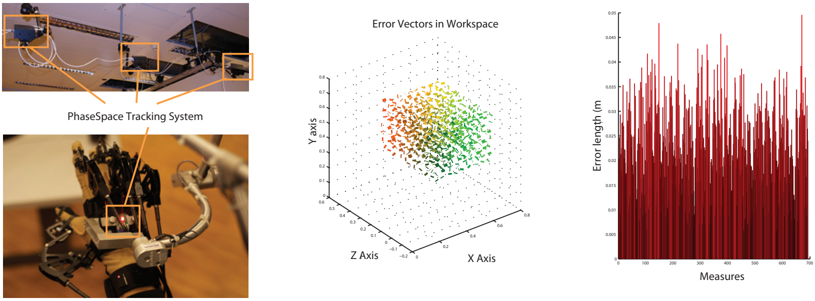


Figure 4.5: Results of the CyberTrack calibration procedure, compared to motion capture data.

In practice, the results are satisfactory for our needs. The registration procedure is really fast, and we can achieve a *praying hands* posture even if is not easy with the four CyberGrasp and CyberForce exoskeletons. In this case virtual and real hands matches.

4.2.3 CyberForce calibration

In section 4.3, we will present a method for reducing user's fatigue when working with the Haptic Workstation^{TM}. The main reason is that the weight of the exoskeletons is not completely compensated by the counterweights. We also show that this extra-weight is not constant inside the workspace, implying that a value of force magnitude on the CyberForce will not have always the same effect on the wrist. The *user's comfort improvement module*, made for that purpose has also the advantage to normalize the vertical force components within the workspace. It means that a vertical force command x will apply a force of y N on the wrist, whatever the position of the wrist in the workspace. Thus, we can consider that a part of the problem is solved for the vertical component. We have to check the two

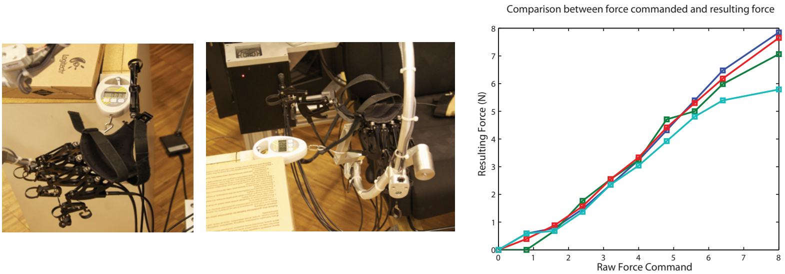


Figure 4.6: Calibration of the CyberForce.

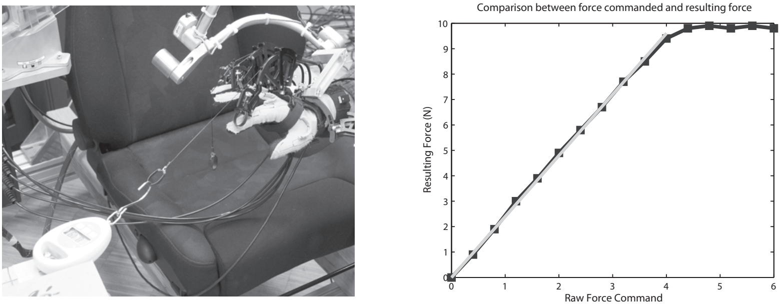


Figure 4.7: Calibration of the CyberGrasp

other components (forward/backward, and left/right).

The second goal of the calibration is the unit of the force commands. The CyberTrack now returns positions in meters (unit of length in *SI*). We would like to have a force command expressed in Newton units. Thus, we only need to calculate the constant converting force commands to SI units. This is done only once, because there is no drift possible at this stage (except possibly motor weariness resulting in loss of torque).

We choose to use a spring scale linked to the ground on one side and to the end effector of the CyberForce on the other side, as shown on figure 4.6. Then, we apply a force and measure the impact on the spring scale. The graphic on figure 4.6 shows four curves: two of them present a set of measures at a particular position of the exoskeleton for the *left/right* force component. On curve is a comparison for the *forward/backward* component, and the last one for the *up* component. We are concerned with two problems. First, we want to know if the force magnitude is dependent of the position of the exoskeleton. Then, we want to find the conversion of the values sent to the CyberForce in Newton.

This graph do not present all the tests made, but the ones shown are representative of all others. We can conclude that the force magnitude is not dependant of the exoskeleton position for the *up/right* component when activating the user's comfort improvement module. It seems to be normal. However, we are slightly surprised that it is also the case for the *left/right* and *forward/backward* components. The second constatation is that when performing a linear regression on the curves, the slope is an approximation of a number allowing us to express force command values in Newtons.

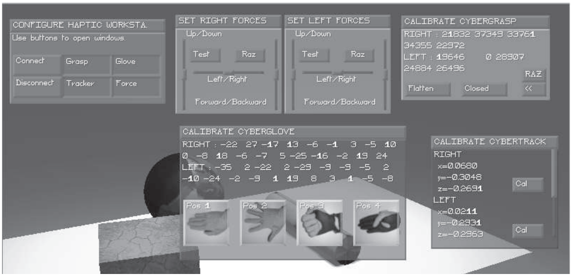
4.2.4 CyberGrasp calibration

In this subsection, we are focusing on the CyberGrasp. The experimental protocol is exactly the same than for the CyberForce: we fix the CyberGrasp and apply force commands on the strings. Results are presented on figure 4.7.

Results obtained shows that the force command is quite linear. The slope at the beginning is around 2.4 and that a maximal force of 10 N could be achieved with this device.

4.2.5 User Interface

The calibration of the CyberForce has to be done at each startup of the Haptic Workstation^{TM}, whereas each time the haptic application user changes, it is preferable to perform a new CyberGlove calibration. In order to avoid to launch another program for performing a new calibration, we propose to take advantage of the MVisio 2D GUI rendering functionnalities to embed in the MHaptic library some GUI elements for the calibration and the connection of the Haptic Workstation^{TM}. The application programmer simply needs to call a single MHaptic function to add the complete GUI in is application. It results in the interface presented on figure 4.8.



```
RIGHT : 21832 37349 33761
34355 22372
LEFT : 19646     0 28907
24884 26496
```

```
RIGHT : -22  27 -17  13  -6  -1   3  -5  10
0  -8  18  -6  -7   5 -25 -16  -2  19  24
LEFT : -35   2 -22   2 -29  -9  -9  -5   2
-10 -24  -2  -9   1  19   8   3   1  -5  -8
```

```
RIGHT
 x=0.0680
 y=-0.3048
 z=-0.2691
LEFT
 x=0.0211
 y=-0.2931
 z=-0.2963
```

Figure 4.8: GUI for connecting and calibrating the Haptic Workstation^{TM}.

There is one main window with 6 buttons: one for connecting and one for disconnecting the Haptic Workstation^{TM}, and four buttons for opening other windows. These windows allow a rapid calibration, provide a debugging information of the data, or allow to manually set forces. This convenient method is really helpful in many situations.

4.3 User Comfort Improvement

Our first experiences with the Haptic Workstation^{TM} has shown that this device is uncomfortable to use during long sessions. The main reason is the uncomfortable posture of the arms, which must be kept outstretched horizontally while supporting the weight of the CyberForce exoskeleton. To prevent this comfort problem, we implemented a solution that is used with all the work presented in this thesis.

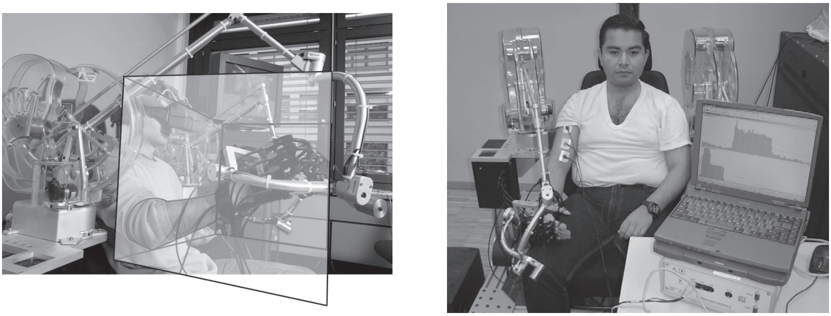


Figure 4.9: A user in the Haptic Workstation^{TM} and the measures of the extra weight of the exoskeleton

4.3.1 Identification of uncomfort

Improving comfort when using a Haptic Workstation^{TM} is not like improving it inside a sofa or a car. It is more like enhancing the comfort of a diving-suit! Indeed, the Haptic Workstation^{TM} was not designed with comfort in mind. After some hands-on experience it is easy to realize how cumbersome and heavy it can be. Using the workstation as shown on figure 4.9, is equivalent to keeping arms outstretched while holding a 600*g* weight on each wrist. A user is not able to stay inside more than half an hour.

The notion of comfort is very close to absence of tiredness: if the users were not tired, they could stay probably longer, and we would consider this as a comfortable device. Tiredness is due to muscular activity (which induces fatigue). Thus, to improve the user comfort, we should at least focus on reducing the muscular activity induced by using the Haptic Workstation^{TM}.

4.3.2 Evaluation of comfort

“To Improve user comfort”: This assertion implies that we need to measure the comfort. Unfortunately, this is a subjective notion which is difficult to quantify. But, as mentioned in the previous section, comfort is closely link to “muscular activity”.

In the rest of this section, we will consider muscular fatigue as an indicator of tiredness and of comfort. The question is how to measure muscular activity over the time. Such measures and experiments are studied and analyzed by physical therapists using bio-feedback techniques. Bio-feedback is a treatment technique in which people are trained to improve their health by using signals from their own bodies [97]. Thus, bio-feedback instrumentation can be used to obtain objective measures of muscular activity and effort, which may serve to deduce the amount of fatigue.

Bio-feedback measures are achieved using many instruments. The most common is the

Electromyograph (EMG). It detects the electrical potential generated by muscle cells when these cells contract, and also when the cells are at rest. This measure is our indicator of muscular activity. The device we are using is a Physio Recorder S [101], shown in Figure 4.9. Its EMG sensor is composed of three electrodes which are used to measure action potentials of one muscle. EMG signals are the manifestation of electrical stimulations, which motor units receive from the Central nervous System, and indicate the activation level of motor units associated with muscle contractions. EMG can be used as a reliable measure of muscular activity and hence is a good indicator of fatigue and comfort. EMG signals are frequently used for providing control commands for prothetic arms [25, 105]. They have also been exploited in studies on user comfort and performance [30].

We use this equipment to evaluate the muscular activity of the biceps and the triceps of the user while using the Haptic Workstation$^{\text{TM}}$. In order evaluate the comfort improvement module, we need to have reference values for comparing the results. Thus, we asked a user to place his arms in two specific postures during 30 seconds:

- The arms relaxed, with the hands laid down on his knees. This posture would correspond to the minimal biceps activity.
- The arms outstretched horizontally. This posture is considered as uncomfortable, since it requires additional effort on the muscles.

Figure 4.10 presents the results of measuring muscular activity on user's arm (biceps) in the cases described before. The measured signal increases significantly when posture changes from resting to activity, the voltage is multiplied by up to five times. The graph shows two curves. The light-colored one represents the muscular effort over time while carrying the exoskeleton, whereas the dark-colored one corresponds to tests without it. The 600*g* extra weight from the exoskeleton must be compensated by the arm muscles, increasing their activity. This is confirmed by the EMG measures: the light curve is above the dark one, indicating an increase of muscular fatigue.

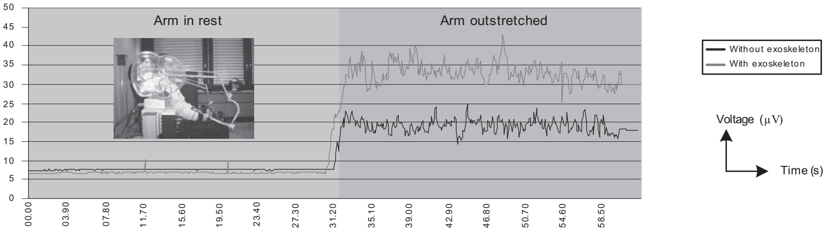


Figure 4.10: Posture and corresponding biceps activity while using the Haptic Workstation

To put in a nutshell, we assume that the muscular fatigue is mainly due to the extra weight imposed by the exoskeleton of the haptic interface. Thus, our goal is to zero the weight of it, which will traduce into improved comfort and longer sessions. Next subsection describes the approach we have followed to achieve this goal.

4.3.3 Anti-Gravity Software

4.3.3.1 Problem Statement

The exoskeletons of the CyberForce^{TM} are equipped with mechanical counterweights. However, they are not sufficient to give the impression that the exoskeleton does not weight. Moreover, depending on its spatial configuration, the exoskeleton do not weight the same as it is shown on figure 4.9. Thus, applying a constant upward force is not sufficient. The force should be applied according to the exoskeleton configuration.

4.3.3.2 Solution principle

The idea is to use the CyberForce itself to counter its own weight. For this purpose we need the upward force values which compensate the effect of the gravity on it for each spatial configurations. These values are stored in a *force field*.

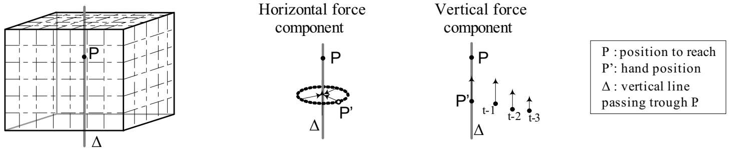
The *force field* is a continuous function. We have two solutions to represent it: the analytic method, which gives exact values of the force for each position after computation; and the discreet method, which returns approximative values and which is based on an interpolation.

However, the equation of the force field based on the rigid bodies of the exoskeleton is hard to calculate and to solve in real-time. In [31], the authors present a comparison of different algorithm for gravity compensation of parallel mechanisms (a 5DOF haptic interface). They did not present a real-time controller for such algorithms. The possibility of evaluating these kinds of compensation functions in real-time is not guaranteed. Moreover this method supposes that we have the weight distribution of every exoskeleton bones. This is almost impossible to measure. These reasons convinced us to choose a discrete approximation method: it is easier to implement, faster to evaluate and sufficiently precise for our needs.

The Anti-Gravity software (AGS) could be divided into two parts: We first need to calculate the force field for each exoskeleton and then to compute the appropriate force value according to the position in real-time. The next two subsections describes these procedures.

4.3.3.3 Force field computation

The first step is to define a working area, a parallelepiped in front of the Haptic Workstation, where the weight compensation would be applied. The compensation depends on the exoskeleton's position, but also on the orientation. However, to simplify the procedure we compute a force field that is only function of the position. As the force vector to be applied is always vertical (it counterweights the gravitation), we can only store the magnitude of the force. The result of this computation is thus a 3D field of scalar values and is evaluated using the algorithm that we describe in the next paragraph.



At each iteration, horizontal force component constrains exoskeleton along Δ. Vertical force component, which is the value searched, is refined to reach the position P .

Figure 4.11: Forces constraining arm position while force field sampling.

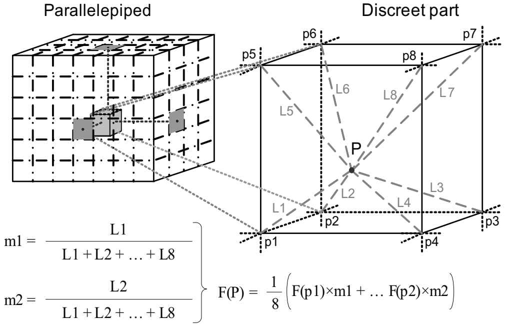
The parallelepiped -working area- is discretized into voxels and a force is calculated at each of their vertices. To find the correct force value, the algorithm is based on a single loop where the vertical force is refined step after step. At each iteration horizontal forces constrain the position of the exoskeleton to achieve each pre-defined vertex, as shown in figure 4.11. There are three conditions to get out of this loop: the exoskeleton must be immobile, the position must be reached, and these two conditions must stay true during at least one second. In fact, immobile means “moving slower than a ε-speed” (less than $0.5mm/s$), and position reached means “closer than a ε-distance” (less than $1mm$). Inside the loop, changes are made to the force according the current state, as shown on table 4.2. When all positions have been reached, results are saved in a XML file.One of the advantage of this method is that the user does not need to interact with the Haptic Workstation$^{\text{TM}}$during the sampling of the force field.

		Moving State		
		Going down	**No Move**	**Going Up**
Position	**Above**	No change	Low decrease	High decrease
	Same pos.	Low increase	No change	Low decrease
	Below	High increase	Low increase	No change

Table 4.2: Force modification in function of exoskeleton states (position and speed)

4.3.3.4 Realtime exoskeleton weight compensation

The next step is to use the values stored in this force field to nullify the weight. When the hand is in the parallelepiped, the vertical force to be applied depends on the current position. Our approach is to search for the discretized part of the parallelepiped into which the hand is located. Then a weighted mean of the eight vertices of this part gives the force value as illustrated on figure 4.12. The actualization of the forces is done within a frequency of at least $800Hz$, which corresponds to the Haptic Workstation^{TM} hardware refresh rate. By this way, we could insure more than enough updates of the forces for the user comfort even for quick movements.



$$m1 = \frac{L1}{L1+L2+\ldots+L8}$$

$$m2 = \frac{L2}{L1+L2+\ldots+L8}$$

$$F(P) = \frac{1}{8}\left(F(p1)\times m1 + \ldots\ F(p2)\times m2\right)$$

Where F(x) is force at position x, mα is weight of point pα, and Lα is distance between pα and P.

Figure 4.12: Getting the compensation force at each position of the space

In this section we have described the software implementation of the Anti-Gravity software. The next step is to evaluate its performance and validate whether it is useful to improve user's comfort when interacting with virtual environments.

4.3.4 Tests Results

In order to test the Anti-Gravity Software, we need to calculate the force field at least once, and then to use the bio-feedback device to have an objective measure of its effect on muscular activity.

First of all, we need to define the parallelepiped into which the force field is calculated and its number of subdivisions. Then the process runs automatically without need of user supervision. The CyberForce^{TM} exoskeleton must reach all positions. To avoid damages on the workstation, the speed of the CyberForce^{TM} is slow (few millimeters per second), thus it takes a long time to accomplish the task (at least 2 hours to reach $7 \times 7 \times 7 = 343$ positions for one arm). Considering that there is no human intervention during this step, the time is not really a problem. Once the force field is saved, the easiest way to test it is to place the exoskeleton at many positions into the parallelepiped and to release it gently. The result is that the exoskeleton does not move: exactly the behavior we were expecting. Moreover, when the user touches the exoskeleton he can feel that there is no force constraining the movement. It means the force is minimal. We have tested experimentally that the minimal weight required to break equilibrium (move down the exoskeleton) is of approximately $2g$.

The bio-feedback device allows to quantify the muscular effort. Thus we have designed a test protocol for the AGS where the user must keep his arm outstretched with and without it.

The results are presented on figure 4.13. The test session is divided into three thirty-seconds-steps. During the first step we activate AGS and the user keeps his arm in the parallelepiped where force field is applied. Then, during the next thirty seconds, we deactivate it and the user has to keep his arm in the same posture. And finally the last step is the reference measure that we will compare to the first one: it corresponds to the posture with the arm outstretched and without the exoskeleton. Peaks that appear at around thirty seconds and one minute should not be considered because they correspond to the movement of the user changing the posture of his arm.

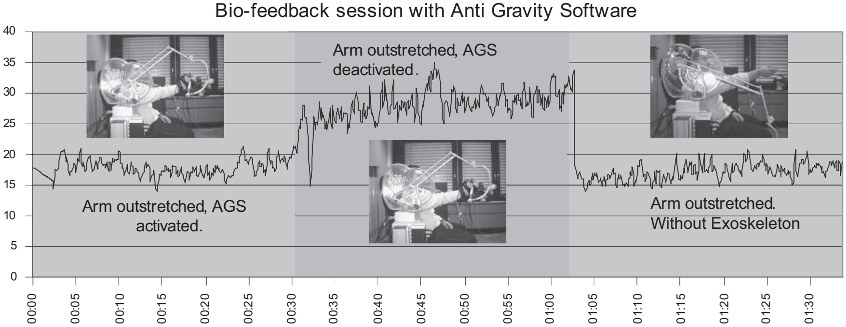


Figure 4.13: Effect of gravity compensation on muscular activity

Values returned by the EMG during this test seem coherent. The reference measure (last $30s$) was already calculated and presented on figure 4.10. We remark that there are almost same values (between $15\mu V$ and $20\mu V$).

We can remark that user's muscular effort with weight compensation is equivalent to the muscular effort for keeping arm outstretched. It means that the AGS compensates for the exoskeleton weight, simulating the earth's gravity, and thus improving user comfort without constraining motion.

In the rest of this thesis, the User Comfort Improvement module could be considered as activated by default in every applications presented (unless mentioned).

4.4 Summary

In this chapter, we presented a study on the steps for correctly registering a two-handed haptic device into Virtual Environments. The first step is to have access to the meaningful data of the device. We provide this access at a high refresh rate near 1 kHz, which is mandatory for efficient haptic interaction. Then, taking into account the possible applications of such devices, we provide efficient calibration tools. Efficiency in terms of calibration is application-dependant. A large-scale two-handed haptic device do not require the same absolute precision than a laparoscopy simulation tool. Our objective is to have calibration methods that are fast, easy to do, keeping in mind that the user is not always the same, and sufficiently precise for the need of manipulation. Finally, with the same state of mind, and considering applications such as teleoperation or virtual training, we believe that comfort is essential for increasing the efficiency. Thus, we provided a method for reducing user fatigue when using the Haptic Workstation$^{\text{TM}}$ [89].

Chapter 5

Realistic Two-Handed Haptic Manipulation

A TWO-HANDED haptic device is a complex device. It has a lot of inputs/outputs, and thus, controlling it requires many computer resources. Moreover, the computation should be performed at a high refresh rate, which is not simplifying the task. Hopefully, today's computers usually include multi-core processors. In this context, we strongly believe that a multithreaded approach is almost mandatory to achieve our goals.

In this chapter, we propose a set of haptic rendering functionalities that have been grouped together into the MHaptic framework. First, in section 5.1, we present the general organization of this library, indicating how to maximize the haptic rendering refresh rate and how to compute the force feedback efficiently. In section 5.2, we introduce the collision detection techniques and the implementation of an existing detector into our framework. In section 5.3, we give indications about the computation needed to perform the physical animation of virtual objects, and we present the integration of a physic engine. And finally, in section 5.5 we propose our hand's model for efficiently interacting with the objects of a virtual environment with two hands.

5.1 MHaptic Architecture

It has became a *de facto* standard that haptic feedback generation start to be efficient near 500 Hz, and even that 1 kHz is preferable [49]. A study on the influence of this refresh rate could be found in [19]. When considering the visual feedback, it appears that refresh rates are significantly smaller. Between 24 Hz and 60 Hz depending on the application : movies or realtime computer graphics. Thus, it seams obvious that we have to separate haptic computation from visual display.

Besides this constatation, another reason could convince to separate these processes: in 2008, even the low-costs personal computers are equipped with a multi-core processors. Thus, we believe that we can greatly increase the efficiency by parallelizing our haptic-based Virtual Reality applications.

In the following subsections, we present the different modules, their role, how they are organized together, and how they communicate.

5.1.1 Modules Organization

The MHaptic library embeds several components. The figure 5.1 presents the general organization of the modules, and shows how they exchange information together. In this section, we explain this diagram, and justify our choice about software organization.

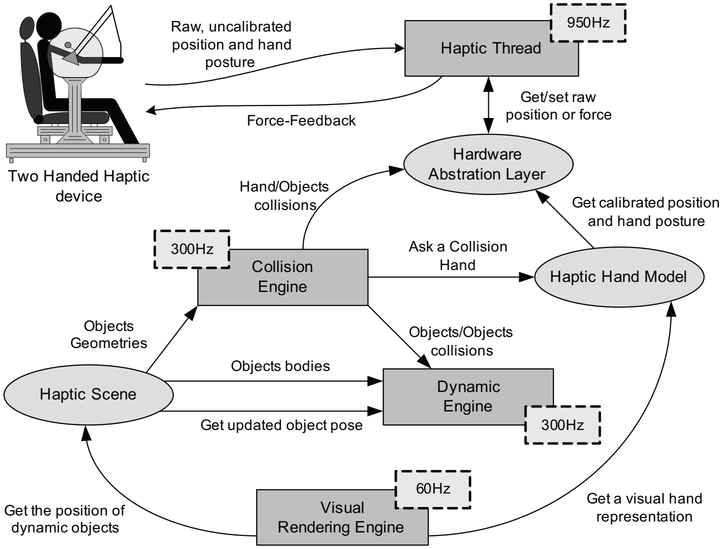


Figure 5.1: General Organization of the MHaptic library.

On this diagram, the modules are presented in rectangles: the *Haptic Thread*, the *Collision Engine*, the *Dynamic Engine*, and the *Visualization Engine*. Each module is in fact an infinite loop, running until the application ends. The oval are representing the data structures that simplifies the communication between the modules. There is the *Hardware Abstraction Layer*, that guaranties the consistency of the data coming and going to the *Haptic Thread* (see section 5.1.2). The *Haptic Hand Model* is one of the most important, as it manage the virtual interacting hands. It will be particularly detailed in section 5.5. And finally, the *Haptic Scene* groups the objects of the Virtual Environment that are touchable and manipulable. The arrows presents the main messages and exchange of information between these components (modules and data structure).

The *Haptic Thread* embeds the optimizations made in section 4.1 for increasing the refresh rate of the data coming and going to the Haptic Workstation^{TM}. Basically, it gathers raw input data (hand posture and position) and copies it into a shared part of the memory. It gives also the force values to the output device. The forces result more or less from the computation of collisions between the hands and the objects. However, these modules do not have a sufficient refresh rate. Thus, we need to update force magnitude even if we do not have an updated collision information. Consequently, the only solution is to make it the *Haptic Thread*. We deal with the force computation in section 5.5.2. Moreover, at this stage, if activated, there is the *User Comfort Improvement* function presented in section 4.3 that can add a vertical force component on the CyberForce.

The *Collision Engine* and *Dynamic Engine* are working together. It seems difficult to parallelize them because one (the *Dynamic Engine*) needs the results of the other. They have many roles. First, they detect the collisions between the hands and the virtual objects. This is essential for the force computation and for user's feeling. They also animate the grasped or touched virtual objects, which is essential for interaction. Finally, they allow to have a fully dynamic environment where each object is able to act on another, which increases realism and possibly immersion.

The last module is the *Visual Rendering Engine*. Its role is of course to provide a view of the Virtual Environment. Haptics without visual feedback is often used in the context of object recognition. But, as mentioned in [48], the discrimination of shapes or curvature using a whole-hand (multi-fingered) kinesthetic haptic device is as difficult than with single-point haptic device. Moreover, a two-handed haptic device seems adapted to virtual manipulation or training. This suggest that embedding a visual rendering system into a haptic framework presents advantages.

As presented on 5.1 these modules are not running at the same refresh rate. The are separated into three threads. The first one contains only the *Haptic Thread* and has a fixed refresh rate around 950 Hz. We remind that this module contains itself two subthreads. The second one embeds the *Collision and Dynamic Engine*. Its refresh rate can be easily adjusted according to the complexity of the Virtual Environment. And finally the *Visual Rendering Engine*, is in fact not running in a thread, but in the main program which is instantiating the other threads. In this state, the MHaptic library could possibly occupies four processor/cores.

5.1.2 Modules synchronization

A well-known difficulty when dealing with multitasking is the memory coherence. It appears, for example, when a thread/process is writing data and that another one is reading the same data. In MHaptic this situation can possibly appear twice. First, when one of the module needs input from the haptic device, or has to command a force. For this issue, we propose to create a *Hardware Abstraction Layer* presented in the next subsection. And the other situation appears when the *Visual Rendering Engine* needs to display an updated position of the scene. We deal with this situation in subsection 5.1.2.2.

5.1.2.1 Hardware Abstraction Layer

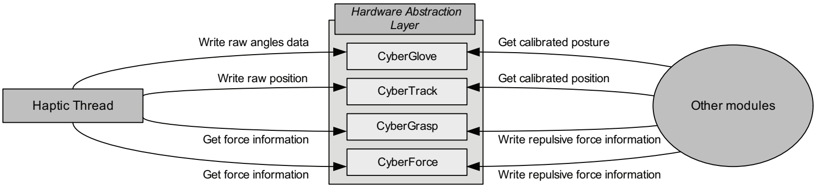


Figure 5.2: Functions of the *Hardware Abstraction Layer*.

To maintain data integrity in a multithreaded haptic framework, we propose to use a *Hardware Abstraction Layer*. Usually, in Computer Systems, the role of such layer is to hide hardware from the rest of the code by providing higher-level functions for controlling the lower-level components. In our case, it expose every functionalities of a two-handed haptic device to the rest of the framework. It means also that MHaptic is designed to manage other devices than the Haptic Workstation^{TM}. If we want to use the framework with another haptic device, we simply need to adapt the *Haptic Thread*.

As shown on figure 5.2, the *Hardware Abstraction Layer* contains four components. Each components has a memory buffer for stocking their related data. They also provides two functions: a modifier and an accessor. There are mutually excluded using a semaphore. It means that they cannot be executed simultaneously on the same component. Concerning the input devices –i.e. *dataglove* and CyberTrack–, the modifier could only be used by the *Haptic Thread* to store raw data into the buffer, whereas the accessor is used by the others modules (for updating the virtual hand model for example). The code of the accessor contains also the conversion from raw to calibrated values presented in section 4.2. Concerning the output devices –i.e. CyberForce and CyberGrasp–, the accessors of the buffer are used by the collision detection system which provides useful data for computing force feedback like contact location, penetration distance or contact normal. And opposingly, the modifiers are used by the *Haptic Thread* to retrieve this information.

Moreover the preservation of data integrity, this method presents another advantage: the conversion from raw to calibrated data is only done when it is needed. Indeed, the other modules do not run as fast as the haptic thread and they need calibrated values less frequently that the *Haptic Thread* is able to update them.

The *Hardware Abstraction Layer* eases the implementation of other functions related to the hardware. For example, in the Cybertrack class, the *Haptic Thread* does not simply store the last raw position in buffer, but it keeps a trace of the 1000 last positions with a timestamp. It does not add extra computation to do so. However, doing so allows us to provide a function that can compute the mean speed or acceleration over a small time period (one second or less). In our implementation, we also use a second-order Backward Difference Expansion (BDE) Estimator to find the instant velocity [23]. Here again, the computation of this kind of information is only performed if it is needed by a component, and thus, do not slow down the *Haptic Thread*. This is in fact the dual of the mechanism that computes the force feedback: the other modules are not fast enough for refreshing the force feedback, so they only provide the elements needed for this computation, and the *haptic thread* performs this computation according to the lastly updated values.

In this subsection, we have presented a simple but powerful data structure inspired by realtime operating systems mechanisms. It eases and secures the communication between hardware and the software.

5.1.2.2 The Haptic Node

In this subsection, we present the *Haptic Node* and *Haptic Scene*, that are two helpers whose primary role is to maintain integrity between the visually displayed information, the collision geometries and the dynamic bodies. As these objects are not being used at the same refresh rate, an effort has been made to guarantee that a module (visual, collision or dynamic engine) always get the lastly updated value.

In Computer Graphics, a *Scene Graph* is a commonly used data structure that arranges the logical and spatial representation of a graphical scene [106]. Usually, it is based on a graph or tree structure linking *nodes* together. It means that a *node* could have many children but a single parent. When an operation like a translation is applied on a *node*, it propagates its effect to all its children.

On one hand, the MVisio visual rendering engine takes advantage and organizes the objects into a *Scene Graph*. But these objects contains only visual properties and could not be efficiently used by our haptic rendering system. On the other hand, we have the two *Collision Engine* and *Dynamic Engine* modules that are using specific information. The two Scene Graphs of these modules are different and could not be changed. Obviously, there is a need for data structure linking these two scene graphs. This is the role of the *Haptic Scene* and *Haptic nodes*.

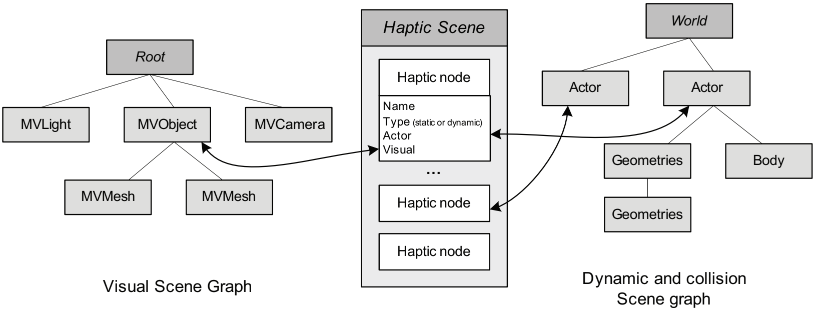


Figure 5.3: The *Haptic Node* links visual meshes and dynamic actors

As presented on figure 5.3, a MVisio scene graph contains a root, and then different kinds of objects organized in a tree. For instance, only the *MVMesh* objects should be be considered because they can potentially represent a tangible object. On the other side, we have a dynamic world that contains object that do not have a truly visual representation. The role of the haptic node is to provide a link between these two scene graphs. Such link is needed to ease the maintenance of coherence between the two *Scene Graphs*. For example, if an actor is moved by the *Dynamic Engine*, the corresponding visual object should be positioned accordingly. For this reason, a *Haptic node* is a class that contains pointers on a visual mesh and on its dynamic alter ego. This class provides also useful methods described below. The *Haptic Scene* is simply a collection of nodes, organized in a simple list.

In a common haptic application, the dynamic world has a refresh rate that is much higher that the visual rendering. Thus, it is not necessary to move a visual object at each step of the dynamic animation. Moreover, only the dynamic actors move. For this reason, every *Haptic Node* declared dynamic is inserted at the beginning of the list, whereas the static ones are inserted at the end. The *Haptic Scene* has a function that is supposed to be called before the new visual rendering of the scene. This functions starts by locking the *Dynamic and Collision Scene Graph*, then it parses the list of *Haptic Nodes* until it finds a static node. Each time it finds a dynamic node it updates the position of the corresponding visual object.

The *Haptic Scene* provides also another interesting functionality. It is able to serialize and deserialize the content of the *Haptic Nodes* using a XML file. During the serialization process, that reads the XML file, a validation is made for guarantying that it is well-formed. This validation is made according to a DTD (Document Type Definition). The choice of XML has been made in order to ease the exchange of data in the haptic community. For this reason, we also created a XSL (XML Stylesheet) document that can converts the data from our XML format to the COLLADA standard. Examples of XML files, the DTD and the XSL files can be found in *Appendix I: The Haptic Scene file format*, page 137.

In the previous sections and chapters, according to the figure 5.1, we have presented the *Two-handed Haptic Device*, the *Haptic Thread*, the *Hardware Abstraction Layer* and the *Haptic Scene*. We will know detail the *Collision Engine*, the *Dynamic Engine* and the *Haptic Hand Model*.

5.2 Collision detection

Although our knowledge of the laws of Physics has evolved along time, the Newton's law of non-penetration, which states that two bodies cannot occupy the same place in space within the same time interval is still not refuted, at least in which concerns the opinion of most people. A physical constraint avoids objects of our world to penetrate each other, and we are used to that. It is essentially because of this fact, that the problem of collision detection came into the Virtual Reality problematic: virtual objects are also expected not to penetrate each other. To achieve a realistic system, we have to take into account this fact.

Collision detection is a broad topic dealing with a quite simple problem. It consists in determining if two or more objects are intersecting. Result of this computation is thus a boolean. However, additionally to the *if (...there is a collision)* question, we may also want to know *when* and *where* the collision happens in order to properly adapt the force feedback response.

In this section, we will first briefly review the computation needed to perform the detection of collisions in Virtual Reality and Computer Graphics. Then we will deal with our implementation of this functionality.

5.2.1 Basics

As mentioned earlier, collision detection is a broad topic. It is even difficult to establish a classification of different methods. Some methods are exclusively applicable to rigid bodies while others can be also extended to deformable bodies. Collision detection methods also depend on the technique used for objects modeling. Methods made for polygonal objects cannot be applied on objects constructed by CSG, for example. In addition, some methods for collision detection are intimately related to collision avoidance methods, some are able to detect self-collisions, others are not. In the context of building a manipulation engine, we will first focus on rigid bodies. Deformable objects (fabrics, liquids, etc.) are much more difficult to manage because they react differently to the collisions.

In terms of collision detection, a rigid body is described by its geometry. In this subsection, we will deal at first with computing intersections between two objects, and then present methods for efficiently retrieving every collisions within a scene using some optimizations necessary to achieve real-time.

5.2.1.1 Intersection of two geometries

Basically, detecting collisions means checking the geometries of the target objects for interpenetration. Mathematics and Geometry give enough tools for that, what is usually called static interference test. We can extract two types of methods :

- **Distance Calculation**. This kind of algorithm start by determining the closest elements of each objects (usually located on face, edge or vertex). This can be done by using brute force or stochastic (Monte-Carlo) methods for example. Then the Euclidean distance between both objects can be calculated, and a negative distance indicates a collision or a contact. [81]
- **Intersection Tests**. Intersection tests are efficient when the geometry or the kind of primitive is known. Given two spheres $A(C_A, r_A)$ and $B(C_B, r_B)$, an intersection exists if $|C_A - C_B| < r_A + r_B$. But usually, a Virtual Environment is not only composed by spheres, and Computer Graphics provides mainly triangle meshes. Triangle-triangle intersection test can also be done efficiently [82], but in the case of complex objets with many triangles, it could require a potentially huge number of tests.

To put in a nutshell, detecting collisions between two primitives (like sphere, box, or capsule) is not really time consuming, but complexity greatly increases when dealing with triangle meshes. But as most of the visual 3D models available are made with triangles and not with the simple cited primitives, there is a strong need detecting collision between general polyhedra.

As pointed out in [78], intersection detection between two complex polyhedra with n and m vertices can be done in linear time in the worst case : $O(nm)$. Using a proper preprocessing, complexity can even drop to $O(\log n \log m)$ [39]. The preprocessing takes $O(n+m)$. However, convex polyhedra is a subset of all possible polyhedra, and handling the general polyhedron case is qualitatively more difficult.

Therefore, instead of managing the general polyhedron, a possible solution is to decompose it into convex parts in order to be able to use the algorithm cited in [39]. This decomposition can also be performed as a preprocessing step. However, the performance of this step depends of the complexity of the preprocessing and of the quality of the resulting decomposition. The minimal decomposition problem is known to be NP-Hard [5]. But earlier, Chazelle proposed an algorithm that can always partition a polyhedra in $O(n^2)$.

In this subsection, we have seen that the efficiency of the inference test between two geometries is strongly dependant of the nature of the geometries, the general polyhedron (triangle mesh) being the worst case. Unfortunately most visual 3D models are made with triangles. Thus, simplifying these models seems to be essential for increasing the performance and the quality of the haptic feedback. We present our method for simplifying the meshes in 6.2.

5.2.1.2 Collision detection in a complex scene

In the previous subsection, we dealt with collision between two given geometries. However, a normal scene is composed by several geometries (n). A naive solution for retrieving every collisions consists in making the static inference test between every couple of geometries: it represents exactly $n \times (n-1) \div 2$ tests, showing a $O(n^2)$ complexity. However, there are usually very few collisions, and testing every pairs is a loss of time. In this section, we presents three basic optimizations aiming at making the static inference test only in the cases that are doubtful.

As shown in the previous subsection, directly testing the geometry of two objects for collision against each other could be expensive. To minimize this cost, object bounding volumes could be tested for overlap before the geometry intersection test. A bounding volume is a single simple volume encapsulating an object that has a more complex nature. The main idea is that this simpler volume has a cheaper overlap test, and that a complex mesh does not collide with another one if their bounding volumes do not collide themselves. The choice of the bounding volume characteristics is important. As shown on figure 5.4, it is a trade-off between intersection test cost and tight fitting of the volume. A large bounding volume could result in a false positive collision and thus, reduce the performances. Our method uses axis-aligned bounding boxes because they are easy to compute, easy to intersect and require less memory. In case of false positive collisions, it does not strongly reduce the performances because we made the choice to approximate the objects with simple primitives (see section 6.2).

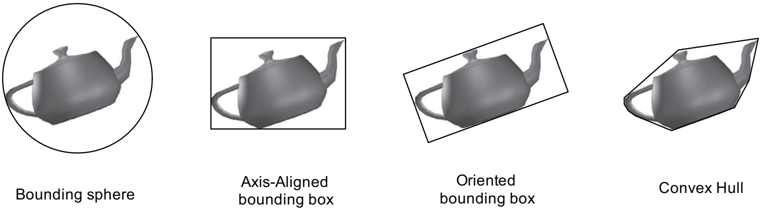


Figure 5.4: Types of bounding volumes

Second optimization is really simple. It comes directly from the fact that, in a normal realistic life scene, only few objects are moving. Thus, every static object that are not colliding a time t could not be colliding at a time $t+\varepsilon$. Thus, during the animation process (which will be described in 5.3), we mark each object that do not move. And then when we parse the pairs of objects, we do not even call the static inference test if they are both disabled. This optimization is very efficient because it replaces the static inference test, which can be complex by two simple bit tests. However, this method do not reduce the number of pairs of object tested.

The third optimization is related to the fact that two relatively small and distant objects could not intersect. So it is not useful to test these kinds of pairs. Two main approaches exists for classifying the objects. First approach consists in organizing the objects in a *Binary Space Partition Tree* (BSP-trees) [85]. This approach is efficient when the scene can be preprocessed. But, in our case, it is not possible because a scene can possibly contain a lot of dynamic objects. Second method is called *Spatial Partitioning*. It consists in dividing the space into regular regions and to test only the objects belonging to the same region. Many techniques exists: uniform grid, *quadtree* [100], *octree* [6], or their generalization, the *k-d tree* [14] [46]. A good study on advantages and disadvantage of these methods can be found in [59].

These combined optimizations greatly increase the performance of the collision detector.

5.2.2 Conclusion

In this section, we first presented techniques for detecting collisions between two geometries. We conclude that simplifying the complex objects into simple geometries saves a lot of computer resources. Then, we have presented optimizations used to speed up the process when dealing with many geometries. These optimizations are needed for achieving realtime performances.

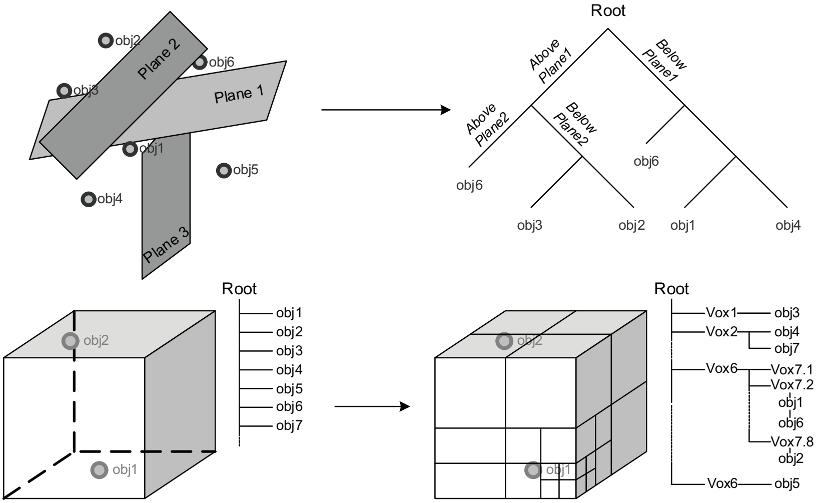


Figure 5.5: A BSP-tree above and Spatial Partitioning technique below.

5.3 Dynamic animation

People are also used to diverse reactions of the objects when submitted to conditions toward the violation of the non-penetration law. Some objects bounce on each other, some deform, some others break; they generally produce sound when colliding, sometimes heat. In Virtual Reality, modeling objects reproducing such phenomena is a big issue on increasing the realism of a scene. Moreover in a context of realistic manipulation, it is absolutely necessary to provide such mechanism. In Physics, the branch of the classical mechanics which is concerned with the motion of bodies is called dynamics. In Computer Animation, we can distinguish two kinds of dynamics:

- Forward Dynamics: the movements are calculated from the forces acting on a body.
- Inverse Dynamics: constraints are applied which specifies how objects interact, for example, they may be linked by a hinge joint or a ball joint, and from this the forces can be calculated.

In our system, we use these two kinds at the same time.

A Virtual Environment is composed by moving objects. In this section we refer to a body when we are dealing with an object with dynamic properties. A body is composed by 3 main parameters:

- The mass m, which is a scalar value.

- The center of gravity G, a 3 dimension position vector relative to the body local frame. Forces acting on a body are applied to this specific point.
- The inertia tensor I, a 3×3 matrix, which represents the angular moment of inertia. More generally it describes how difficult it is to change the angular motion of a body about the main axes.

Forces acting on a body tend to move it (displacement and orientation). To calculate the resulting change of position we use the Newton laws (especially the second one), and the resulting change of orientation could be computed using the Euler's equations of motion. Here are the two laws:

$$\sum F = m\ddot{x_G} \text{ , and } \sum T = I_A \ddot{\alpha} \text{ , where :} \tag{5.1}$$

F and T are the the forces and torque applied on the rigid body.

m, G, I, are the mass, center of gravity, and inertia tensor.

x, α are position and orientation of the rigid body.

To solve the equations 5.1, we can distinguish the approximates methods or the analytic methods. One of the most frequetly used in computer animation is the Euler Integrator which is explained in [7]. In [87], Otaduy and Lin present a good review of these techniques when applied to haptic rendering.

In our framework, the rigid body animation is used in two contexts: the first one is to realistically animate the Virtual Environment. The second one is to implement the hand model.

5.4 Implementation of Physics

In this section, we propose a study on the implementation of the *Collision* and *Dynamic Engines*. A *Physics Engine* is a middleware that combines these two components together. Some years ago, it was necessary to write its own collision detector and to manage the collision response in order to perform rigid bodies animation. But, the research in this field and the increased computational power lead some groups to release publicly available *Physics Engine*. A review of existing *Physics Engine* can be found in [18].

Historically, in MHaptic, two engines were successively implemented: the Open Dynamic Engine (ODE), which is an open-source constraint-based engine that uses a Euler Integrator. The second one is the *NVISIA PhysX*, previously named *Novodex* and owned by *AGEIA*. This one is free to use (even for commercial applications) but sources are not included. ODE has some important features that are difficult to manage. For example, each geometry must have a local frame that is centered on it. And it is impossible to change it. In our context, the problem is increased, because we want to interact and manipulate existing Virtual Environments without modifying them. Thus, we need to perform several coordinate system changes to synchronize the visual and physical representations of a single object. This task is really tedious. Moreover, when very high values of forces and torques are applied on joints, typically when the user tries to squash an object with his hands, they degenerate. This is really annoying because our implementation of the hand uses such joints. Opposingly, joints in *NVIDIA PhysX* do not generate because it is much more numerically stable, and the geometries could have arbitrary coordinate system.

Moreover, every optimizations presented in 5.2.1.2 could be easily integrated with *PhysX*. Thus, it appears that this library provides almost every tools that we wanted in order to implement a framework for two-handed haptic interacting with generic Virtual Environments.

5.5 Haptic Hand Model

In the two previous sections, we presented a way to physically animate the virtual objects. The main advantages of using this method is that when we exert external forces on virtual objects, we do not need to provide specific behaviors (using scripts). But it means also that the virtual hands should be modeled with respect to the same approach.

In this section, we present a powerful model of virtual hands in the context of haptic manipulation.

5.5.1 Approach and implementation

Several approaches have been primarily tested based on different methodologies. In this subsection, we present the two techniques that were implemented and tested.

5.5.1.1 Direct Mapping

It consists in positioning a virtual interaction point at the same position than the device itself (hard link). This is the most trivial solution, and seems also to be the easier to implement (but we will show later that it is finally not the case). The calibrated position, rotation and posture of the hands are directly mapped onto the the virtual hand. The hand model is thus composed with collision geometries and with a skeletal animated visual mesh that uses skinning techniques for increasing realism. The idea behind this technique is to compute collisions between the hands and the virtual objects and to apply a realistic force-feedback for avoiding the collisions.

As presented in 5.2, the collision detector returns *if* and *where* two objects collides. This second information is used to compute the force feedback. For example, if the fingertip enters into a table, two points are returned:

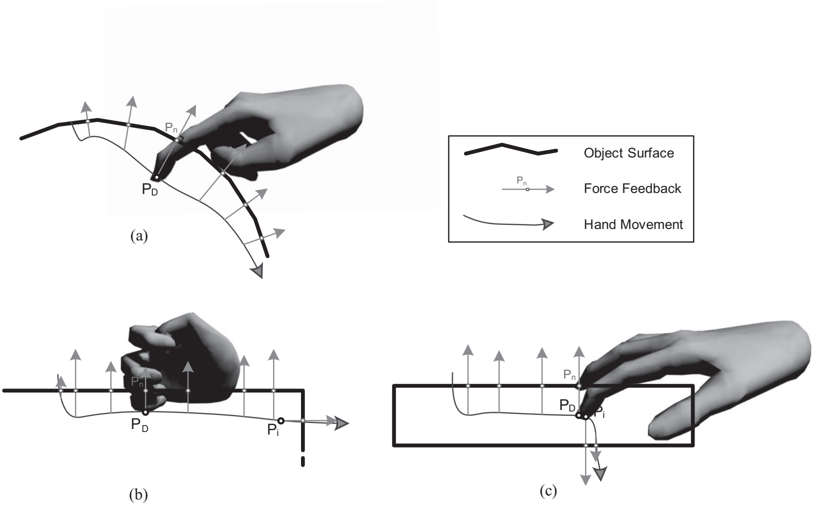


Figure 5.6: Force feedback Computation based on penetration distance

- The deepest point of the phalanx into the table (P_D on the figure 5.6)
- Another point P_n laying on the table surface that is the closest as possible of P_D

The vector $\overrightarrow{P_D P_n}$ determines the direction of the force feedback, and its norm $\Delta = |P_D P_n|$ is used to compute the magnitude of the reaction force. This method gives the impression that an elastic with a zero rest length is link between P_n and the fingertip resulting in a force $\mathbf{F} = -k\Delta$. As for every virtual elastic, it is possible to change the spring constant k. In order to avoid the resonance, due to the fact that the user's fingertip is oscillating between a "inside-object"/"outside-object" state, it is also possible to add a damping factor d for smoothing the force magnitude: $\mathbf{F} = -(k\Delta + d\dot{\Delta})$. In fact, damping is critically important, due to its role in counteracting the energy generation from errors introduced by sensing and discrete time.

The first difficulty with this method is to deal with user's movement. The figure 5.6 presents three particular cases. The first one 5.6(a), shows the resulting force feedback when the user displaces his hand on the surface of a spherical object. We can observe that the force seems to be smooth and the variation of direction gives the impression that the object is a sphere. However, the two other examples presenting a hand moving on the surface of a table (see figure 5.6(b) and 5 5.6(c)) are showing force continuity breaks at particular points P_i. They are due to the fact that the collision detector returns the closest point to P_i laying on the object surface. We can imagine many solutions to avoid this specific problem, but in fact, each solution will present an inconsistency in a particular context.

The main source of problem with this solution comes from the Haptic Workstation itself. The break sensation in force continuity increase with the penetration depth (because force magnitude is greater). Our haptic device is not powerful enough to counterbalance the weight of an arm resting on a table (or barely). Thus, penetration distances can become great and the perception of a break increases.

Another problem to mention which is related to the Haptic Workstation^{TM} appears when the force feedback could be applied on several actuators at the same time. The figure 5.7 presents two cases when it happens. The first one happens shows user's fingers in contact with the border of a table. When the fingers penetrates into the virtual table, a resistive force has to be applied. But, it raises a question:

Should we apply the force feedback on the wrist through the CyberForce^{TM},
or on the fingers through the CyberGrasp^{TM},
or even on a mix of both ?

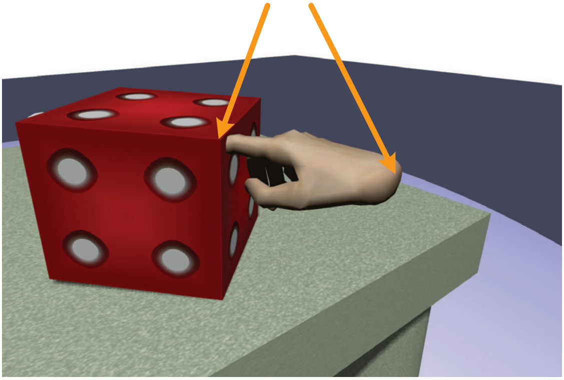


Figure 5.7: Where should we apply the force feedback ?

In fact, it is impossible to answer to this question by examining a single frame at a single time. The reason is that it depends of the muscular tension of the wrist and fingers. If the user locks his wrist and that finger muscles are resting, the force feedback should be mainly sent to the CyberGrasp^{TM}(and opposingly of course). If every muscles rest, force feedback should be applied on the two devices at the same time. The same problem appears when two hands are in contact with the same dynamic object. In this case, it is not trivial to compute the force magnitude on the CyberForce^{TM}, because each hand transmit force through the virtual object to the other hand. As our Haptic Workstation^{TM} is a passive device, we only know the position of the hands, and not the forces applied by the hands on the system. This is the reason why it is impossible to answer to the question using a single state of simulation. However, by examining many consecutive frames, we can get an approximation of hand's speed and acceleration. Then, because of the relation $\sum \overrightarrow{F} = ma$, it is at least possible to approximate the force divided by the mass.

Finally, with the direct mapping method, there are visual inconsistencies due to the interpenetration of hands and object resulting in a break in presence [103]. It does not satisfy the law of non-penetration. Here again, the limited force of the Haptic Workstation$^{\text{TM}}$ is the source of the problem. Moreover, it does not ease the computation of the force feedback, and some specific examples shows that it could introduce discontinuities in force direction. We present another solution in the next paragraph. We choose it for avoiding these two problems.

5.5.1.2 Mass Spring Hand Hand

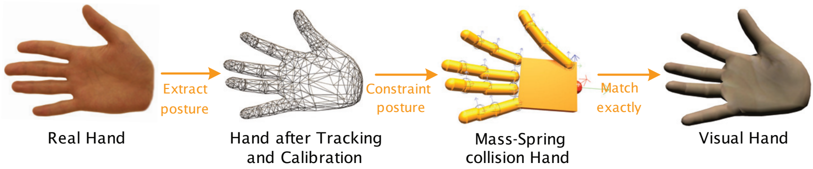


Figure 5.8: The three Hand models

Our second technic consists in using a "god-object" or a proxy-based method [120]. A proxy is weakly linked to the position of the device, i.e the Haptic Workstation$^{\text{TM}}$. Basically, it consists in three hand models (see figure 5.8):

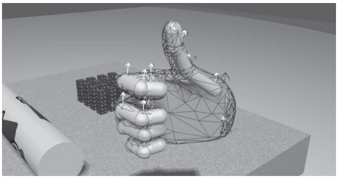
- The Tracked Hand (shown in wireframe on the figure). It is in fact the virtual hand skeleton created after calibration. It is supposed to be the exact representation of the real hand position orientation and posture into the Virtual Environment. It is of course not the case, but we assume that the matching is correct.
- The Proxy (Volumes shown on the figure), which is a mass-spring-damper system that has the shape of the hands. Each phalanx and the palm is composed of a collision geometry, and has also dynamic properties. These elements are linked together with motorized joints parameterized with springs and damping coefficient.
- The Visual Hand. This is the hand that is visually rendered and the only one visible. It is easily created using the MVisio visual rendering engine [94].

For each hand, the idea is to couple a proxy hand to the tracked hand using a set of virtual linear and angular springs. As a result of the dynamic simulation, the spring-hand tends to follow the tracked-hand. The visual hand displayed to the user reflects the spring-hand configuration.

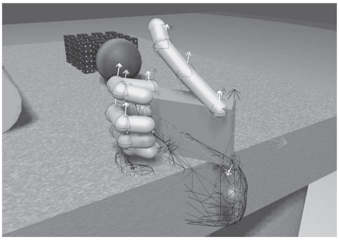
This approach follows the "proxy" method proposed for the Phantom (a single-point interaction device), extending it to the whole hand. It has been firstly described by Borst et al. [20]: they applied it to the CyberGrasp force feedback device which is also a component of the Haptic Workstation$^{\text{TM}}$.

It solves the problem of interpenetration between the visual hands and the environment (mentioned in 5.5.1.1) because the spring-hands adapt their pose on the surfaces of the objects. Spring-Hands have basically two constraints:

- A soft constraint which is to match the best as possible the configuration of the tracked hands. This is achieved by applying specific force and torques on the linear and angular springs.



- A hard constraint which is to avoid penetration within virtual objects. This is achieved simply by activating the collision detection between the phalanxes/palm rigid bodies and the objects of the Virtual Environment.



The spring-hands are created using dynamic rigid bodies. To match the posture of the hands, we use two kinds of geometrical primitives. For each hand, the phalanxes are approximated with capsules and the palm with a flat box. Then, these rigid bodies are linked together using spherical joints (3 angular DOF). Some of them could be replaced by hinge joints (only 1 angular DOF) though we did not notice any problem with this method. By linking the geometries together with those joints, we get an articulated hand. Finally, an angular spring is attached to each joint. These springs will give a torque to the phalanxes according to the angles of the tracked hand. As a result, the fingers of the spring-hand will follow the fingers of the tracked-hand. The torque τ applied by the spring on the articulation made of an angular joint is computed as follow:

$$\tau = k(\alpha_s - \alpha_t) - d(\omega_s - \omega_t) \text{ , where :} \tag{5.2}$$

k, d are the spring and damping of the angular joint,

α_t, α_s are respectively the angles of the tracked hand and of the spring-damper hand,

ω_s, ω_t are angular velocities of the phalanxes on the tracked hand, and on the spring-damper hand

The spring constant defines how stiff the torque is applied to the fingers. An high value will provide a more reactive behavior but will suffer from vibrations. The damping constant allows the torque to be reduced according to the respective angular velocities of the joints. It avoids the vibrations of the fingers but provides a smoother reaction. These two parameters are set empirically. More details can be found in section 6.1.

By the same manner, to allow the translation and the rotation of the hands in the virtual world, one linear spring (3 DOF) and one angular spring (3 DOF) are attached to the base of each hand. Therefore, when the user moves the wrists, these two springs respectively apply a force and a torque, allowing the base of the spring-hand to follow the base of the tracked-hand. The linear spring provides a linear force F as follow:

$$F = k_t(p_t - p_s) - b_t(v_s - v_t) \text{ , where :} \tag{5.3}$$

k_t, b_t are the spring and damping constants,

p_t, p_s are positions of the tracked and spring hands,

v_s, v_t are velocities of the spring and tracked models.

The torque that is applied by the angular spring is more complex as it depends on the direction of the hands. As a result the spring torque vector and the damping torque vector usually do not point in the same direction.

In the *NVIDIA PhysX* library, it is possible to directly attach linear and rotational springs to the joints. The spring constant k and the damping constant b are both parameterizable. According to the SDK documentation, it is preferable to adopt this solution than computing the forces and torques and apply them manually to the phalanxes. The reason is that it may be quickly instable for stiff spring and forces due to a limitation of the numerical integration used to advance time from one time step to the next. Internally, the joints use a drive constraint in which the springs are implicitly integrated within the solver. This is the best way to model stiff behavior for stable simulation. We followed this recommendation as the spring constants are necessarily very high to ensure that the spring-hands follow the user's moves as close as possible.

In fact, it implies that the forces and the torques are not applied directly as presented previously. Instead, we give angular orders to the drives (motors) of the joints and the *Physics Engine* integrates these values to internally produce the forces and the torques. However, the model presented here is still perfectly valid in our case and represents the real behavior of our simulation.

In the previous paragraphs, we have exposed the concepts and techniques related to the Mass Spring Hand. However, we did not clearly present its implementation. The figure 5.9 schematizes the data structure, and present also the standard pipeline going from calibrated haptic data to visual hand model.

To put in a nutshell, the Haptic Hand Model is a data structure which contains three different hands. Internally, the tracked hand is stored as a hierarchy of positioning matrices.

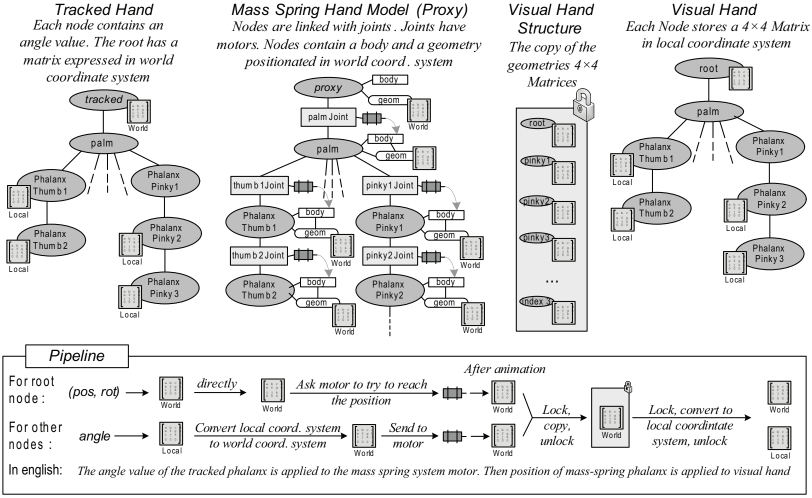

Tracked Hand
Each node contains an angle value. The root has a matrix expressed in world coordinate system
tracked
World
palm
Phalanx Thumb 1
Local
Phalanx Thumb 2
Local
Phalanx Pinky 1
Phalanx Pinky 2
Local
Phalanx Pinky 3
Local
Mass Spring Hand Model (Proxy)
Nodes are linked with joints. Joints have motors. Nodes contain a body and a geometry positionated in world coord. system
proxy
body
geom
World
palm Joint
palm
body
geom
thumb 1 Joint
pinky 1 Joint
Phalanx Thumb 1
Phalanx Pinky 1
thumb 2 Joint
pinky 2 Joint
Phalanx Thumb 2
Phalanx Pinky 2
Visual Hand Structure
The copy of the geometries 4×4 Matrices
root
pinky 1
pinky 2
pinky 3
...
index 3
Visual Hand
Each Node stores a 4×4 Matrix in local coordinate system
root
palm
Phalanx Thumb 1
Phalanx Thumb 2
Phalanx Pinky 1
Phalanx Pinky 2
Phalanx Pinky 3
Local
Pipeline
For root node:
(pos, rot)
directly
Ask motor to try to reach the position
After animation
For other nodes:
angle
Convert local coord. system to world coord. system
Send to motor
Lock, copy, unlock
Lock, convert to local coordintate system, unlock
World
Local
In english: The angle value of the tracked phalanx is applied to the mass spring system motor. Then position of mass-spring phalanx is applied to visual hand

Figure 5.9: The *Haptic Hand Model*, and its embedded data structures

Then, the Proxy hand, which is the most complex, is stored as a hierarchy of *PhysX* actors (an actor is a combination of one or more geometries and a body). The links of the hierarchy are 6DOF joints (with distance constraints and angular limits). And each joint has a motor linked to it. The motor acts mainly on the second body of the joint. Finally, the visual hand model is also stored as a matrix hierarchy, combined with an MVISIO skinning mesh.

As shown on the figure 5.1, two modules need an access to the *Haptic Hand Model*, the first one being the *Collision Engine*, while the second one is the *Visual Rendering Engine*. As these modules run in different threads and need the same data, we have to synchronize them. We chose to perform this synchronization at the visual hand model level, by mutually excluding the positioning matrices stored in the *visual hand structure* of the figure 5.9.

Thus we provide two functions: the *"Hand Model Update" Function* is called by the thread containing the *Physics Engine*. We present it in the Algorithm 1, and we can see that it follows strictly the pipeline of the figure 5.9. In this algorithm, we can notice that the *visu* variable contains in fact a simple copy of the matrices positioning the geometries. Thus, the last part of the algorithm is executed quickly, and the lock time on *visu* is not long.

On the other side, the *"Get a Visual Hand Model" Function* gets a copy of *visu*, then converts the matrices that are in the world coordinate system into matrices in local coordinate system (based on the hierarchy), and finally applies it directly to the bones of the MVISIO hand model.

In this section 5.5.1, we have presented two approaches that have been chronologically

Algorithm 1 Hand Model Update

```
Require: $track$ : A tracked Hand Model
Require: $proxy$ : A Mass Spring Hand Model
Require: $visu$ : A Visual Hand Data Structure
  {Initialization of local variables. This is a synchronized access managed by the }
  $pos \leftarrow$ Calibrated Hand Position
  $rot \leftarrow$ Calibrated Hand Rotation
  $angles \leftarrow$ Calibrated Gloves Angles List

  {First, we convert the angles directly to the proxy model}
  $track.pos = CreatePositioningMatrix(pos, rot)$
  $track.pinky3.matrix = CreateAroundXRotationMatrix(angles[0])$
  $track.pinky2.matrix = CreateAroundXRotationMatrix(angles[1])$
  ... {And so on for every angles}

  {Then, we send the orders to the mass spring hand motors}
  $proxy.pos = track.pos$
  $proxy.motorPinky1To2.reachWorldPosition(track.pinky2.getWorldMatrix())$
  ... {And so on for every motor}

  {Finally, we store the data for easily creating the visual model. It is also a synchronized
  access}
  Lock $visu$
  $visu.pos = proxy.pos$
  $visu.pinkydistal.matrix = proxy.pinkyproximal.geom.matrix$
  ... {And so on for every visual phalanx}
  Unlock $visu$
```

implemented. The first one shows many disadvantages that convinced us to use the second solution. In the next sections, we discuss the computation of the force feedback using the mass spring hand.

5.5.2 Force feedback Computation

The mass spring hand model provides an elegant way to compute the force feedback. When a collision with the hand occurs, it is indeed easily possible to compute the distance between the position of the tracked hand and the position of the mass-spring system. Then, we send these two positions to the *Hardware Abstraction Layer* (see subsection 5.1.2.1), especially in the CyberForce and CyberGrasp data structure. Then, the *Haptic Thread* get these data at a high refresh rate, and is able to compute a force for making the *tracked hand* to match the position of the *mass-spring hand*. The force magnitude is of course proportional to the distance between the two models. On the figure 5.10 (page 71), we can see the difference between the two positions of the tracked and mass spring hands. The resulting forces are represented by the arrows. The advantage of the method is that the forces are refreshed according to the lastly updated positions, near 1 KHz, whatever the speed of the collision engine. The disadvantage is that during the real hand movement we can miss some collisions. This occurs only with small objects: for a hand speed of $1ms^{-1}$, and at 300 Hz, the detector checks collisions every 3 mm. This is the concept of force feedback computation.

However, the previous paragraph did not mention some details. The CyberGrasp is an unilateral device. Thus some collisions could not be managed. To solve this problem, when we compute the difference vector between the tracked and mass spring hands, we also get the vector that is normal to the nail (the nail's normal vector is aligned with the string of the Cybergrasp). Then, if the scalar product of these two vectors (after normalization) is positive, it means that the CyberGrasp should produce force.

The *Collision and Dynamic Engine* are running in the same thread. It is their results that are sent to the *Hardware Abstraction Layer*. The algorithm 2 resumes what they are doing. On the other side, the *Haptic Thread* gets these results and computes the forces. This is presented for the CyberForce$^{\text{TM}}$ in the algorithm 3. In this algorithm, *k* is a variable that takes into account the conversion to Newtons found in 4.2.3 and that can vary over time to perform some kind of low/high-pass filtering.

We mention also that using the same method we are able to compute the torque. In this case, when using Euler angles, the computation of the (minimal) angle between two rotations is less straightforward than the distance between two positions. But when using quaternions to represent the rotations, it is immediate. Unfortunately, because the CyberForce is an underactuated device, we never tested it.

5.5.3 Benefits for manipulation using two-hands

In this section, we will show that the mass-spring hands presents many advantages in the context of two-handed haptic interaction.

The *first advantage* concerns the visual feedback. We have seen that when using this system, the visual hand do not penetrate anymore into the virtual objects. However, it induces a problem often called visual-proprioceptive discrepancy. This problem occurs when the virtual hand is for example stopped by a table. It is then possible that the real hand is not at the same position that the virtual one (Position discrepancy). It is also possible that the real hand moves but the visual one does not (Motion discrepancy). This is of course something that the user could notice. The question is then to know which anomaly is noticed the first. In [26], Burns et al. investigate users detection threshold for visual interpenetration and visual-proprioceptive discrepancy. Conclusions show that we are much more sensitive to interpenetrations. We can thus conclude that the spring-hand model present an advantage over the direct mapping technique in terms of presence.

The *second benefit* that we have presented concerns the computation of the force feedback. In the case of interaction through a single-point device using for example a Phantom®,

Algorithm 2 Collision detector and Dynamic Engine Thread

```
Require: $track$ : A tracked Hand Model
Require: $proxy$ : A Mass Spring Hand Model
  while Simulation is Running do
    Call Hand Model Update Function (Algo. 1) {First, we need to update the Hand model}
    $listLeft \leftarrow$ Collision of Left Hand with Geometries
    $listRight \leftarrow$ Collision of Right Hand with Geometries
    $listObj \leftarrow$ Collision of Geometries with Geometries
    Call Animate every objects {including the hands...}

    for all $collision$ in $listLeft$ do
      {We check which part of the Left Hand collides}
      if $collision.geom1 = proxy.thumb1.geom$ or $collision.geom1 = proxy.thumb2.geom$ then
        {We have a collision with the left thumb, we compute the distance vector}
        vector $dist \leftarrow (collision.geom1.pos - track.thumb2.pos)$
        vector $norm \leftarrow$ the normal to the thumb's nail
        if $dist \cdot norm \geq 0.0$ then
          $CyberGrasp.setForceThumb(proxy.thumb2.geom.pos, track.thumb2.pos, dist \cdot norm)$
        end if
      else if $collision.geom1 = proxy.index1.geom$ ... then
        ...
        ... {We perform the same for each finger}
        ...
      else if $collision.geom1 = proxy.palm.geom$ then
        $CyberForce.setForce(proxy.palm.geom.pos, track.palm.pos)$
      end if
    end for

    for all $collision$ in $listRight$ do
      $contacts[6] = $ true;
      ... {Same computation with Right Hand}
    end for
  end while
```

Algorithm 3 Haptic Thread CyberForce Computation

```
while Haptic thread is Running do
  $(pos, rot)$ ← Raw data coming directly from the two-handed device
  {We store the updated position in the Hardware Abstraction Layer}
  $CyberTrack.setPosition(pos, rot)$
  $pos2 \leftarrow CyberForce.getMassSpringPos()$
  if $\|pos2 - pos\| \geq \varepsilon$ then
    $sendForce(\|pos2 - pos\| \times k)$
  end if
end while
```

the force feedback computation is greatly improved due to the proxy. As mentioned earlier, it removes the strange behavior that usually happens when we compute forces according to the penetration distance.

Finally, this method has also a *third advantage*. It implicitly computes the forces applied by the user on the virtual object, as illustrated on figure 5.11.

These improvements strongly increases the realism and thus the immersion of the user into the virtual environment. We try to simulate the best as possible the reality and its physic. This implies a better learning curve of the manipulation system, and requires less adaptation from the user. The efficiency of many applications is improved because users could focus on the simulated task only, and not on the way to manage the simulation itself.

5.6 Summary

In this chapter, we presented MHaptic. It is a software library for performing efficient haptic interaction and virtual manipulation with a two-handed device. We gave at first a general organization of it, showing that parallelization is a promising technique for creating such library, because the different components do not have the same refresh rate. Moreover, we showed how to optimize the transfer of data between these components, without compromising the integrity. Then we explained the needs in term of collision detection and realistic animation engine. We found that the NVIDIA PhysX *Physics Engine* was efficient according to these needs. And finally, we proposed a powerful hand model that allows realistic manipulation with two hands combined with a convincing force feeling.

However, even if this library has a powerful architecture and provides state of the art algorithms to interact with some virtual objects, it is not enough to tell that we achieved the interaction with *any* Virtual Environment. One of the main reason is that usually a Virtual Environment do not include object properties necessary to their animation or to the computation of the force feedback. Starting from this constatation, we propose to add to the MHaptic library a tool that can be used to easily add such properties to an existing visual Virtual Environment.

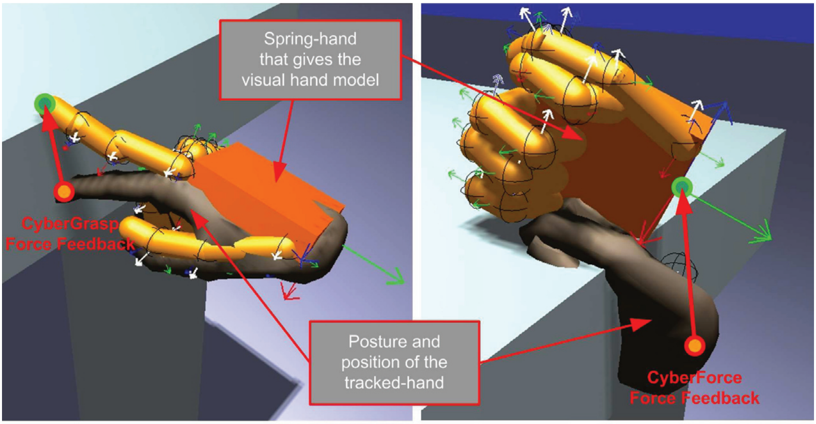


Figure 5.10: The computation of the forces for the CyberForce^{TM}, and CyberGrasp^{TM}

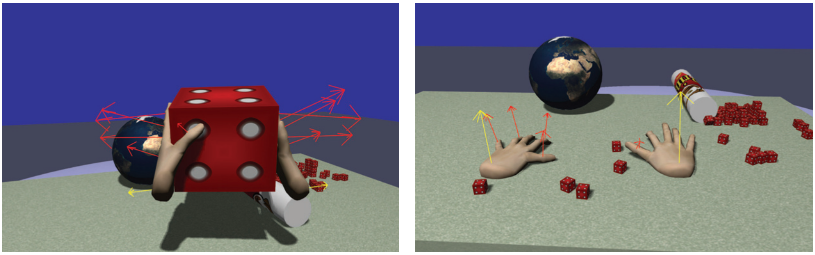


Figure 5.11: The implicit force estimation process done by the mass-spring system

Chapter 6

Manipulation in Generic Virtual Environments

WHEN DEALING WITH two-handed virtual interaction using a haptic device, the ultimate goal is to be able to manipulate and interact with any kinds of Virtual Environments as if they were real. It is of course impossible. We are still far to visually simulate the reality despite all the efforts and the progress in visual rendering during the two last decades. But we can however consider that a high level of realism has been achieved (shown on figure 6.1). In Haptics, the evolution is much more slower. The device has limits that are difficult to handle, and these limits are not only due to computer sciences and electronics, but mainly to mechanics.

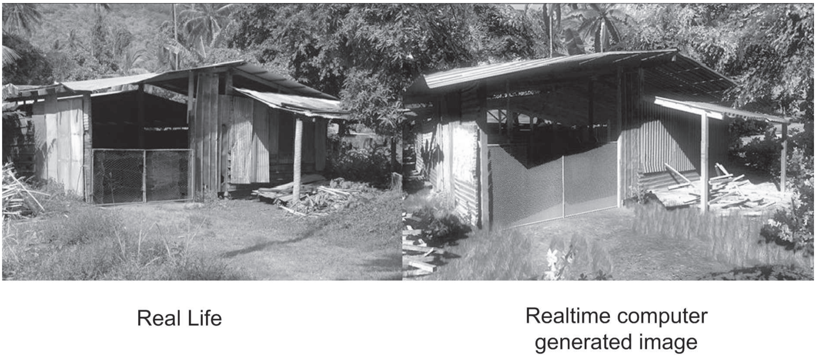


Figure 6.1: Real Scene on the left and Realtime Rendered image on the right using the CryEngine 2 made by Crytek.

Our main goal is to interact with a Virtual Environment using the two hands via the Haptic Workstation^{TM}, to feel the forces exerted by the virtual objects, and to minimize the learning curve of a given application. We also want to ease the work of the application programmer in the sense that he should be able to quickly build a Virtual Environment that has the properties mentioned.

In this chapter, we present some ways to achieve these two goals. The first section deals with the parametrization of the mass-spring model of the virtual hands, while the second section presents an authoring tool intended to add haptic information to the objects of an existing visual Virtual Environment.

6.1 Parametrization of springs, damper and surfaces

In the mass-spring system presented in section 5.5.1.2, it is possible to parameterize many coefficients presented in the following list. These parameters are important because they can alter the quality of the simulation. In this section, we present an analysis for tuning these parameters according to the kind of application.

Here is a list of these parameters:

- The Coulomb friction coefficients and the bounciness of the geometries constituting the hand.
- The Coulomb friction coefficients and the bounciness of the Virtual objects.
- The mass of the palm and phalanxes.
- The spring and damping constant of the phalanxes joints.
- The spring and damping constant of the 3DOF translational wrist joint.
- The spring and damping constant of the 3DOF rotational wrist joint.

We examine at first the friction and restitution of the Virtual Environment objects and of the hand's masses. We then study the parametrization of the springs a dampers of the phalanx joints.

6.1.1 The Coulomb Friction and the Restitution

In this subsection, we present three values that are important because they can greatly alter the difficulty of the manipulation. When hands are in contact with an object, they exert a force on it. But, as in reality, the forces are not sufficient to grasp object. Indeed, the friction has a really important role which is to avoid that the object slides. Friction and restitution are coefficients that need to be tuned for increasing the realism of the manipulation.

Friction is the force resisting the relative motion of two surfaces that are in contact. When two contacting surfaces move relative to each other, the friction between these objects converts the kinetic energy into heat. This effect prevents objects from sliding as

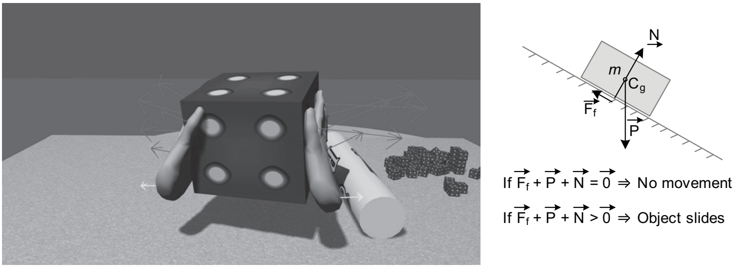

N
m
C_g
$\vec{F_f}$
$\vec{P}$
If $\vec{F_f} + \vec{P} + \vec{N} = \vec{0}$ ⇒ No movement
If $\vec{F_f} + \vec{P} + \vec{N} > \vec{0}$ ⇒ Object slides

Figure 6.2: Friction helps us to grasp objects

shown on figure 6.2. In our framework, we should be able to manipulate the objects in the virtual environment easily. Thus, we should prevent that objects slide though the hands, and opposingly, we should prevent that they stick.

The Coulomb friction is a model to describe friction forces using the equation 6.1. It is in fact an approximation of what happens in reality and not an exact representation of the real physical interactions. The coefficient of friction is a dimensionless positive quantity and could not be calculated: it has to be set empirically or measured experimentally and depends on the two objects that are in contact.

$$F_f = \mu F_n \text{ , where :} \tag{6.1}$$

F_f is the force exerted by the friction,

F_n is the normal force exerted between the surfaces,

μ is the coefficient of friction, either static or dynamic friction depending on the relative movement.

We can distinguish two friction factors: the dynamic friction, and the static friction. In the example of the figure 6.2, the initial minimal force to get the cube moving is modulated by the static friction factor. When the cube is already sliding, the dynamic friction represents the force resisting to the movement. Usually the dynamic friction coefficient value is smaller than the static coefficient.

The restitution (bounciness) is a coefficient that represents the ratio of the speed of a moving object, from when it hits a given surface to when it leaves the surface.

In the physics library used, these coefficient are assigned on a per-object basis. This does not reflect reality, because both friction and restitution are really properties of pairs of materials rather than of single materials. However, providing values for every possible combination of materials that may touch is probably not feasible because it is proportional to $O(n^2)$, where n is the number of materials. Instead, an effective value for the property in question is generated whenever two objects of different materials interact by using a function to combine the two coefficients. This function can be the average, the minimum, the maximum, the addition or the multiplication.

We choose to have interactions between virtual objects (not including the hands) that are not completely correct. Thus, we are empirically setting the friction coefficients of the materials according to standard tables [112], and then we perform a mean of the two materials that are in contact. It creates some errors. However with this method, we obtain results that are visually convincing, due to the fact that it is difficult for a human to estimate friction coefficient using only visual cues. Thus, it is difficult to perceive an incoherence between visually estimated friction and the knowledge of virtual object's material. Concerning the restitution coefficient, we are also combining values using the average function.

A particular attention should be taken to set the coefficients of hand's geometries. The main problem comes from the static and dynamic friction. We found indeed that a small value for the restitution (for instance, 0.1) is appropriate. Because it avoids that the object bounces on the hands, it increases the ability to catch it. However, we distinguish two cases for the friction coefficients: the first one appears when the user touches a static object (usually heavy or fixed), whereas the second one happens when the user tries to grasp a small dynamic object. In the first case, if the object is adhesive, the user should feel resistive forces on the hands. This is automatically computed with the mass-spring hand model, but we have to set a hand value of friction that is quite high to simulate this effect. On the other hand, if the object is slippery, the feeling should be smooth, an this implies a low value of friction. Thus our best results are achieved by setting a medium value for static friction, and a smaller value for dynamic friction. The effect is quite convincing, but still not perfect. Hopefully, such values works well for the grasping of dynamic objects.

6.1.2 The Mass Spring Damper model of the Hand

As previously explained, our hand model is made of masses linked together with joints. First, our approach consists in choosing the masses of the phalanxes, and then to tune the damping and spring parameter of the joints according to the masses.

According to Kreighbaum et al. in 1983 [71], the hand represents an average of 0.5% of the total female weight, and 0.65% of the male weight. Men's mean weight is around 80 kg resulting in hand's weight of approximatively 500 g. For women, these values give hand's weight of 330 g. Although we provide in our haptic library the functionality to change this, we focus in our study on the men hand's weight. All the phalanxes represents 30% of the hand's weight. Thus, we instantiate the palm geometry with a mass of 350 g, the five phalanxes linked to the palm weighting 13 g (metacarpal), then 10 g for the five next (proximal), and finally 9 g for the four last phalanxes (distal).

The spring and damping vales of the joints are probably the most important values. It is important to understand their respective functions. Springs are used to have a reactive hand model that follows quickly the real hand. When springs are soft, a quick movement of the hand could result in a high position difference between hand model and real hand. Whereas damping is used to reduce springs force according to the magnitude of the force.

Its most important effect is to limit the oscillations that could be induced by high values of spring.

We made empirical experiences in order to find the best values of spring and damping. The experiences consist in analyzing the behavior of the spring hand model with various couple of spring/damping coefficients. We start with wrist link, then continue by tuning the parameters for every phalanxes at the same time. Starting with small values, we change them according to user's perception. When he notices oscillations, we increase the damping, and if he notices a movement delay, we increase the spring. With this method, the values are quite different depending on the user. However, we also remark that the ratio between spring and damping is nearly the same for all users. This result is not really surprising and comes probably from the way we adjust the parameters. What is more surprising is the absolute difference between two users. We found indeed that small coefficient values suitable for a user A, are not really good for a user B, and opposingly.

In this section, we presented a study for parameterizing object surface properties in the Virtual Environment and the mass-spring hand model. With these parameters, it is possible to achieve an easy manipulation of different kinds of virtual objects that has been simplified into basic collision detection primitives (capsule, sphere, box, etc.). Concerning more complex meshes, the manipulation is still possible but unexpected behaviors could unfortunately appear. For this reason, we propose in the next section a tool intended to simplify the complex meshes into basic primitives.

6.2 Parametrization of the Haptic Virtual Environment

When creating various applications that have an extensive use of haptic features, the programmers (creators of the application) often face an important problem: the lack of haptic information of the 3D models. The common editing tools for creating Virtual Environments do not always provide the adequate functionalities for adding this kind of information. And when the functions are present, they are barely used by the designers themselves for many reasons. The **first reason** is that the model was not primarily supposed to be integrated with an haptic application. This is often the case for old purely visual models. The **second reason** is that there are many kinds of haptic information that could be added, but most of them are completely irrelevant for a particular haptic device. For example, the temperature of a virtual object will not be used in an application that uses only the Haptic Workstation^{TM}. Moreover, in the common visual design process, after the model creation, there are many passes of prerendering followed by model refinement, etc. It is thus possible to imagine that it should be the same for the *haptic design*. And in this case, it implies that the designer has access to the haptic device by the same manner that he has access to a visual rendering device (usually his screen). This is not always possible because of the costs or the location of the device. It is the **third reason**.

In this context, it appears to be necessary to give the opportunity to the haptic programmer to augment the visual Virtual Environments using an authoring tool, called in the rest of this document the **Haptic Scene Creator**. This one is presented in this section.

6.2.1 Needs and specifications

As previously stated in 5.2.1.1, the complexity of a visual mesh requires its decomposition in low level primitives in order to speed up the collision detection. Obviously, this decomposition can not be done automatically, because it strongly depends on the targeted application and desired level of detail. For this purpose, the Haptic Scene Creator application is a graphical tool that supports the haptic augmentation of any Virtual Environment loadable in our visual rendering engine. It is very intuitive, simple to learn and provides useful automated features to simplify and accelerate the task.

In order to improve the learning curve of the Haptic Scene Creator, we have decided to imitate the kind of interface proposed to advanced 3D graphics modeler such as Autodesk® 3DS Max and Maya (see figure 6.3). It means that our editor has a GUI and is able to load `.mve` visual files, to load/save haptic XML files, and to present a view of the Virtual Envrionment to the designer. Finally, as in Autodesk® 3DS Max, we choose to use a four viewports interface and to place a toolbar on the right of the screen.

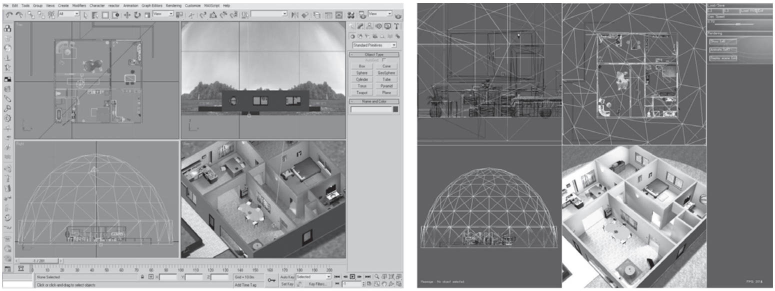


Figure 6.3: Comparison between Autodesk® 3DS Max user interface on the left and the HSC user interface on the right

The designer should be able to select visual objects in order to “augment” them. Once selected, the user can manipulate information which is relevant for our *dynamic engine* and *collision detection* system.

The *dynamic engine* needs information related to the “mass” of an object. We remind that the “mass”, or body, includes the real mass (or density), the center of gravity and the inertia tensor. Of course the mass/density is easily parameterizable in the editor, but we preferred to hide the inertia tensor to the designer to keep the tool intuitive and accessible. In fact, the center of gravity and the tensor matrix can be computed knowing the size, the position and the density of the geometries linked to the object. Unless some special effects are desired, it provides a reasonable approximation. By automating these computations, the designer will also be able to focus on higher level tasks.

The *collision detection* system needs to know the shape of an object. Of course, the shape is clearly defined in its visual description. However, using this information only is by far too complex for the computation of the collisions [42]. Thus, objects should be approximated with simple geometric primitives (boxes, spheres, capsules). Later on we add more complex geometries such as convex meshes and penetration maps. These geometries are not displayed during the simulation, but it is necessary to visualize them in the editor. Moreover, for each collision geometry linked to an objet, its material properties must be parameterizable. These parameters are the static and dynamic friction coefficients and the coefficient of restitution (bounciness) as described in subsection 6.1.1.

Finally, as in Maya animation tool, the immediate visualization of the results using the simulation mode allows the designer to see in real-time the animated objects. This features does not allow the designer to immediately feel and manipulate the objects using the Haptic Workstation$^{\mathrm{TM}}$, but it is more intended for debugging than for fine tuning of parameters. For this specific purpose, it is preferable to launch the real simulation. Now that the specifications of the Haptic Scene Creator are clearly described, we show their implementation in the next subsection.

6.2.2 Implementation of the GUI

We have seen previously, that our GUI needs five main features:

- Open/Save files functionalities.
- Multiple viewport rendering.
- Geometry/material editing.
- Mass property parameterizing.
- Immediate dynamic simulation previewing.

The core of the application uses a Finite State Machine (FSM) shown in figure 6.4. Most operations, commands or drawing executions depends on the current state. State machines provide a powerful mechanism to avoid multiple tests. For instance, it is authorized to modify the mass of an object only if it has been selected, i.e. if the state is “Object Selected”. Internal commands are also triggered during a state transition, such as the synchronization of all views on one object when the user just picked one. Some components of the 2D and 3D GUI are also displayed according to the state of the application. The 2D GUI contains for example the toolbar (see figure 6.5), information messages, the selection area and so on. The 3D GUI contains some objects integrated to the scene that belong to the editor, such as the arcball.

The Haptic Scene Creator is implemented in C++, using only the MHaptic and MVisio libraries.

Some basic features of the Haptic Scene Creator are presented in the following subsections, whereas some more complicated/automated features that have been progressively added are described in section 6.2.3.

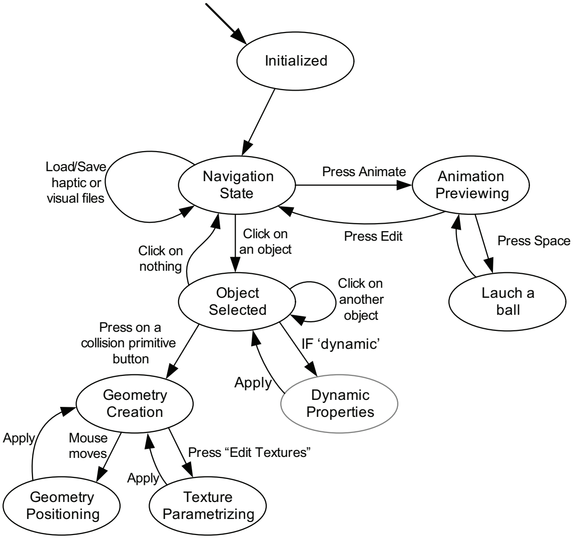


Figure 6.4: The Finite State Machine is the core of the Haptic Scene Creator

6.2.2.1 Multiple viewports rendering and navigation

Navigation using more than one viewport is intuitive because it simplifies visualization and editing, especially when dealing with complex sets of data. Most of the Computer-Aided Design (CAD) software provide the ability to display a 3D object from multiples points of view, as shown in figure 6.6.

In the Haptic Scene Creator, three orthographic cameras and one camera in perspective are provided to visualize the environment. In the three orthographic views, the objects are rendered in wireframe mode. The objects in the perspective view are shaded using transparency, textures and smooth lighting.

6.2.2.2 Creating geometry and body of an object

As stated previously, a haptic object contains two parts: the mass properties, also called body, and the collision geometries.

In the Haptic Scene Creator, the body is computed automatically according to the position of its geometries and their respective densities. The only information that the designer may want to modify is the mass or the density of an object. This can be simply done in the toolbar on the right side of the screen.

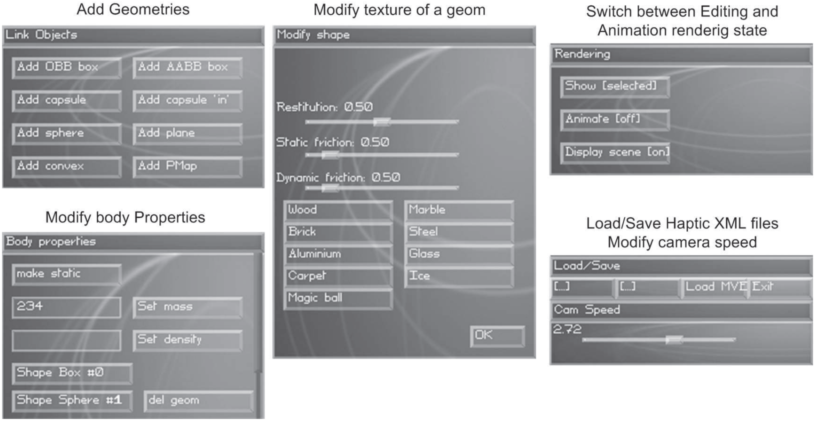


Figure 6.5: The toolbar contains all available commands of the Haptic Scene Creator

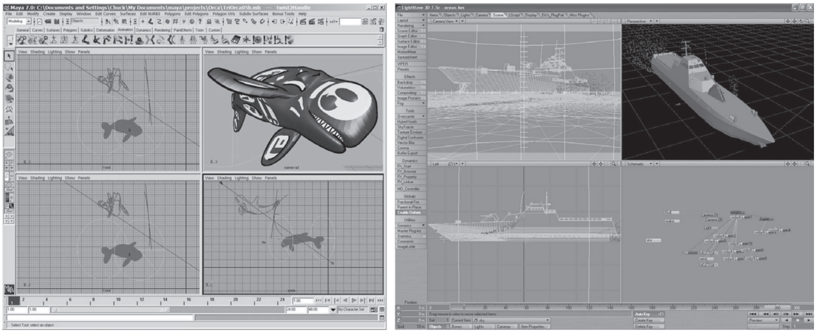


Figure 6.6: Multiple viewports rendering in CAD systems: Maya on the left and LightWave 3D on the right

The designer must also be able to approximate the geometry of a visual object by adding and linking some simple collision geometries to it, such as boxes, spheres, planes and capsules(also called capped cylinder, or line-swept spheres). These primitives can be defined by a low number of parameters: three for a box (height, length, depth), one for a sphere (radius), etc. In the first implementation of the Haptic Scene Creator, the input of these values is performed manually by the designer. However, even if this method works, it is a relatively tedious task. This convinced us to implement a method that is described in section 6.2.3.

6.2.2.3 Editing materials

As introduced previously in subsection 6.1.1, each object has some material properties, which are the restitution (bounciness), the static friction and the dynamic friction.

Materials can be modified for each geometry in the scene. For example, a carpet would provide a stronger frictional force-feedback to the user's hands during its manipulation than a wooden table. The restitution and the frictional coefficients can also be set for each collision geometry independently, allowing various interesting effects.

In the Haptic Scene Creator, the designer can modify materials by using three scrollbars. Some materials have also been predefined in the editor. They are accessible in the toolbar when editing a geometry. One simply has to click on the button corresponding to the predefined material to modify the properties of the selected object.

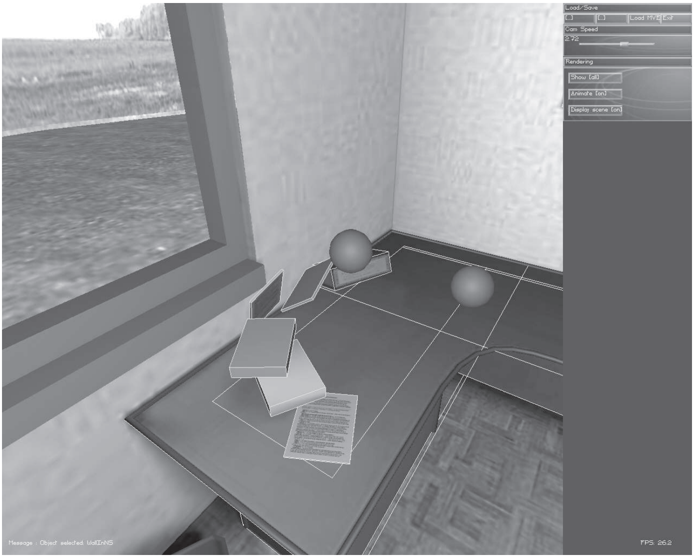
6.2.2.4 Immediate physics simulation of the system

The Haptic Scene Creator provides also an immediate simulation of the augmented haptic scene, as shown in figure 6.7. During the simulation, the designer is able to launch virtual balls on the objects, in order to see if the reaction is correct, and if the collision geometries provides a good approximation.

In this section, the basic features have been covered. However, in this state, the Haptic Scene Creator suffers from a lack of flexibility. At usage, it is not possible to augment quickly a scene. The task is repetitive and tedious. In the next section, we will present advanced features that reduce the time and the difficulty of it.

6.2.3 Advanced functionalities of the Haptic Scene Creator

The basic functionalities presented in the previous section allow us now to start working on a scene augmentation. However, repetitive work and lack of flexibility convinced us to add extra functionalities. These features are implemented in order to maximize speed and comfort during the augmentation of the Virtual Environment. They provide a strong support to the designer automatizing tasks and allowing a fine tuning of the haptic properties.



Message : Object selected: WallInNS

FPS: 26.2

Figure 6.7: The dynamic simulation of the system in the Haptic Scene Creator. The designer can launch balls on the dynamic objects.

The main difficulty when augmenting a virtual environment is to deal with the parametrization of the geometries that approximate a specific object. This is the most time-consuming part. We believe that this process could be partially automatized.

6.2.3.1 Vertex selection and Copy-pasting

Most of the time, the complex objects could be separated into parts that can be easily approximated using simple primitives. Thus it appears necessary to offer the possibility to extract some portions of a single object. We thus provide the possibility to select specific vertices on the selected object, allowing the Principal Component Analysis (described in the next subsection) to be performed only on the subpart of this object.

There are two ways to internally check if a given vertex must be selected or discarded. The first one is to compute the six planes that define the 3D frustum in the world space and to check if the vertex lies in this frustum. The second one is to project the vertex on the screen and check if it lies inside the 2D selection rectangle. The last method is adopted, though both methods give the same results.

We also provide a copy pasting functionality. It is easy to implement and greatly increase the haptic augmentation speed of a complex scene.

6.2.3.2 Fitting collision primitives using Principal Component Analysis

In this subsection, we introduce a method that allows the automatic fitting of the collision geometries to the visual objects. It even allows to fit them to a subpart of an object by using the vertex selection as presented in the previous subsection. Computing tight-fitting bounding volumes is a not a trivial task. Depending on the type of geometry and on its orientation, the difference of volume between a poorly aligned and a well-aligned geometry can be quite large. This can be seen on the middle of the figure 6.8: the axis aligned bounding box is containing all the points but its orientation implies a bigger volume. A better alignment is shown on the right side.

To make this possible, we use the statistical method called Principal Component Analysis (PCA) [67]. In statistics, PCA is a technique for simplifying a multidimensional dataset by extracting its most significant directions (principal components). In our context, the data is the set of 3D vertices of the 3D models. In the Haptic Scene Creator, we use PCA to fit four kinds of shapes: oriented boxes, spheres, capsules and planes.

The procedure is performed in three steps. First, the covariance matrix is computed on the set of selected vertices. Second, the matrix is decomposed to find the eigenvalues and corresponding eigenvectors. Finally, the shapes are oriented according to these eigenvectors. We present now the procedure in detail.

The first step is to compute the covariance matrix $C = [c_{ij}]$ (also called the dispersion matrix) for the selected points $P = \{P_1, P_2, \ldots, P_n\}$.

$$c_{ij} = \frac{1}{n}\sum_{k=1}^{n}(P_{k,i} - u_i)(P_{k,j} - u_j) \tag{6.2}$$

The u_i and u_j terms represent the mean of the i-th coordinate value of the points:

$$u_i = \frac{1}{n}\sum_{k=1}^{n} P_{k,i} \tag{6.3}$$

The second step is to decompose this covariance matrix to extract meaningful information about the principal directions of the point cloud. This decomposition of the matrix is performed by computing its eigenvalues and eigenvectors. In general, finding the eigenvalues and eigenvectors of a matrix in a robust way is nontrivial. However, the iterative Jacobi method can be used when dealing with symmetric matrices, which is always the case for a covariance matrix. As a result, it decomposes into real (not complex) eigenvalues and an orthogonal basis of eigenvectors. In our case, the covariance matrix is always 3x3 sized and thus three eigenvectors are returned after its decomposition. The Jacobi method is not explained here, but more details can be found in [53].

The third and last step depends on the type of shape we are trying to fit to the point cloud. For each kind of shape, we compute its dimensions, position and orientation. The orientation is always computed according to the eigenvectors, because they provide the

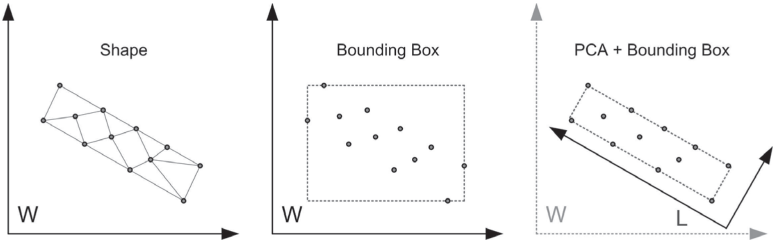


Figure 6.8: PCA applied on an oriented box

main directions of the point cloud. For example, the largest eigenvalue corresponds to the eigenvector indicating the main direction of the point cloud. Finally, the haptic structure of the selected object is updated to include the newly created shape in its local reference frame. We now present how each shape is individually fit.

6.2.3.2.1 Fitting boxes The orientation of the box is set to follow the orientations given by the three eigenvectors of the matrix decomposition. This is shown in two dimensions on the right side of the figure 6.8. The dimensions of the box are then set by first finding the distance between the extreme points of the point cloud along the three axis.

6.2.3.2.2 Fitting capsules As shown on figure 6.9, the capsule is oriented along the axis following the maximal spread of the point cloud. This axis is given by the eigenvector corresponding to the highest eigenvalue found in the covariance matrix decomposition. Then, the height of the capsule is computed by subtracting the minimal and maximal values of the projection of the points along this axis. Finally, the radius of the cylinder is set to the maximal distance between this axis to the furthest vertex in the point cloud. The capsule is then positioned at the center of the cloud.

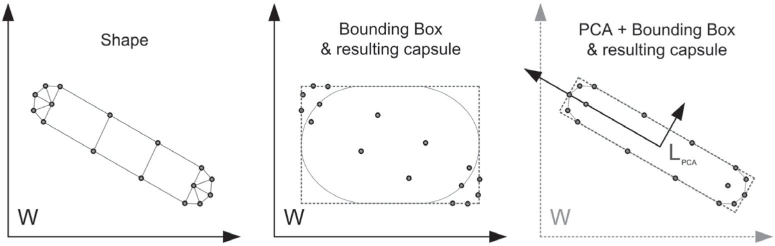


Figure 6.9: PCA applied on a capsule

6.2.3.2.3 Fitting spheres Sphere shapes are defined by the radius only. Fitting spheres is done iteratively. The center of the point cloud is found using the same computation previously done with boxes and capsules. Then, for each vertex not included in the sphere, the radius (initially set to zero) is increased until all vertices finally lie inside the volume.

6.2.3.2.4 Fitting planes Planes are defined by the equation $\overrightarrow{n} \cdot \overrightarrow{x} = d$, where $\overrightarrow{n}$ is the normal of the plane and d its distance to the origin. In mathematics, a plane is a infinitely thin 2-dimensional surface. In many collision detection engines and in our case, planes are considered as half spaces, making an object behind it colliding with it. As a result, the plane is certainly one of the most robust shape found in a collision engine.

Because the plane equation must be computed in world space, the vertices are first transformed in world space. The normal $\overrightarrow{n}$ is set to the unit vector following the eigenvector corresponding to the smallest eigenvalue. The distance d is set to $d = \overrightarrow{n} \cdot \overrightarrow{x}$, where $\overrightarrow{x}$ is a random point from the set of the selected vertices. Because there is no way to know in advance in which direction the normal will be pointing (an horizontal plane can be defined by a vector going up or down), the normal of the plane can be switched manually.

In this subsection, we have presented a method that fits automatically the collision geometries to the visual objects. The results obtained and some comments are given at the end of this chapter.

6.2.3.3 Convex shapes and penetration maps

Adding simple primitives to the haptic structure is sometimes not enough to cover complex 3D models. When a more accurate physical representation of the shape is needed, increasing the number of primitives may become a difficult task. Two kinds of geometries are added to solve this problem: convex hulls and penetration maps. Both geometries are presented in figure 6.10.

Among other uses, convex hull allows a stable approximation of arbitrary triangle meshes. The convex hull of a set of points is the smallest convex set that contains these points. Computing a convex hull is not a trivial task [3] and many algorithms have been proposed [10]. The details are not shown here since the PhysX engine that we use already provides such a robust implementation.

When saving the scene into a file, the vertices of a convex hull are stored in the XML file. This way, the convex shape is only computed once at its creation.

When some concave objects may not be reasonably approximated using standard primitives and convex hulls, the editor allows the use of arbitrary triangle meshes. However, the triangle data is not used directly for collision detection, which could be heavy for such objects. This is why the collision detection engine that we use provides penetration maps (pMaps). PMaps use a voxel representation of the collision geometry instead of the triangles that form its surface.

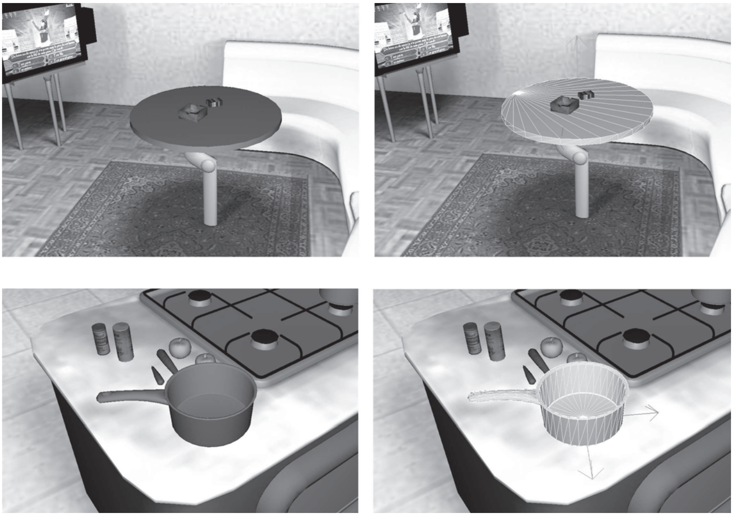


Figure 6.10: A convex hull approximating a table (top) and a penetration map approximating a pan (bottom)

Like convex hulls, pMaps are computed only once at their creation by the collision engine. The voxel data is then serialized in a binary *.pmap* file for a later reuse. PMaps are robust for collision detections but use more memory and are time intensive. We also observed some strange behaviour when building pmaps from complex volumes, so they must be used sparingly. Convex shapes should be used instead as much as possible.

6.2.4 Haptic Scene Creator Evaluation

As shown previously, the Haptic Scene Creator includes many features aiming at simplifying the task of a Haptic application designer who is using the MHaptic framework. To evaluate it, we asked a programmer with knowledge in Virtual Reality to augment a visual Virtual Environment as much as he can. We gave him the the user manual of the software that can be found in the *Appendix III: HSC User Manual*, page 145.

The test scene is a quite complex. It represents a four-rooms house (kitchen, bathroom, bed, office and living-room), and contains 398 visual nodes. It is presented in figure 6.11. The main goal is to use the Haptic Scene Creator to augment every touchable object with a good level of approximation in order to manipulate objects easily. This task took almost three hours, time to create 612 geometries for 167 static objects and 114 dynamic objects. The advanced functionalities (copy-pasting, vertex selection, PCA) were of course exten-

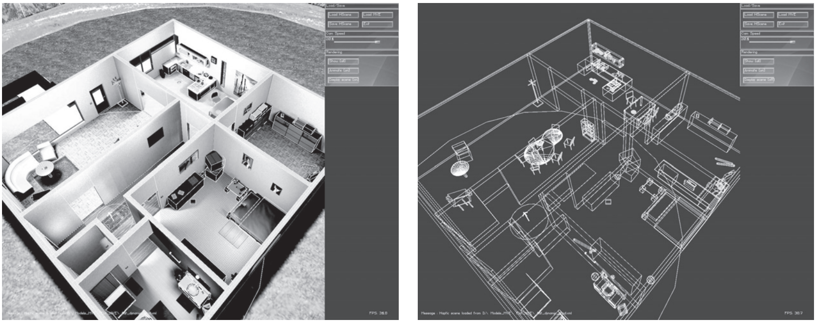


Figure 6.11: The augmentation of the 3D House.

sively used. We do not have performed the same procedure on the full scene without using the advanced functionalities (and neither using a simple text editor to write by hand the XML file). But, to give an idea of the improvements, augmenting a chair without advanced functionalities took five (tedious!) minutes, whereas it took around 1 minute using the PCA.

We remarked that sometimes it is difficult to find the good approximation of an object. It is also sometimes not trivial to find the proper way to select the vertices of a mesh, in order to have the good results of the PCA. But most of the time, a solution can be found. We present an evaluation of the manipulation of this model in section 7.2.2.

Synthesis of Part II

In this chapter we proposed a study on the parametrization of the virtual objects properties. We demonstrate the importance of the parameters for the surface to increase the efficiency of the manipulation tasks. Then, we proposed a tool for easily augmenting the existing visual Virtual Environment by including haptic properties.

The framework that we proposed in the second part of this thesis reach the expected goal: proposing two-handed haptic feedback and virtual manipulation in generic Virtual Environments. We exposed the difficulties and the problems faced during the creation of this framework as well as their solutions. In the next part of this thesis, we present some applications that we built using this framework.

Part III

Applications

Chapter 7

Realistic Manipulation

THE PREVIOUS PARTS of this thesis have presented many algorithms, techniques and innovations related to the control of a two-handed haptic device. We embedded most of this research into a framework: MHaptic. But the primary intention when designing a haptic library (like any other kind of library), is that it can be used by software developers to create at least one application.

The Haptic Interface enables Human Computer communication trough touch, in response to user movements. This communication is **bidirectional** in the sense that the same device conveys information from *Human to Computer*, by the mean of movements, and from *Computer to Human* by the mean of touch or kinesthesia. This characteristic makes the field of Haptics to have numerous applications. Obviously a two-handed haptic device is not thought for a particular application. It can work with every virtual applications that simulates a real life scenario with hand's interaction. But it can also be applied to any context that needs a complex human computer interaction. Considering this flexibility, we believe that we should not bound the evaluation of such two-handed devices to a specific field, as well as we should not make only studies that concerns the realistic manipulation. We will present in this chapter applications with realistic or pseudo-realistic interaction, whereas in chapter 8, we present experiments with interaction paradigms that could not be considered as realistic, because they use metaphors or gestures.

The word **realistic** has many definitions. In this chapter, it is used with the sense of "conform to a reality" or believable. The first section present a feasibility study in the context of virtual training. The objective is to explore the haptic virtual manipulation in a Mixed-Reality Environment. Then, in the second section, we present two applications that uses MHaptic and its Virtual Environments interaction functionalities.

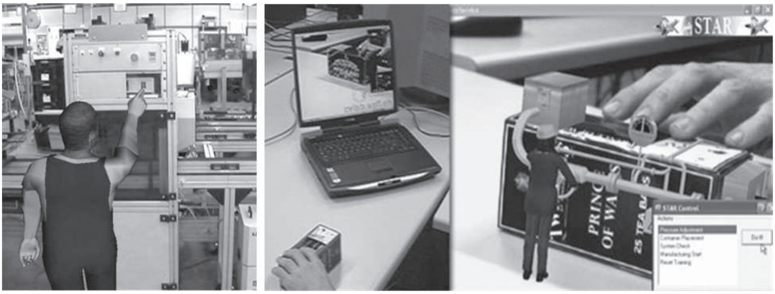


Figure 7.1: A Mixed-Reality industrial training environment [115]

7.1 Two-handed Haptic Manipulation in Mixed-reality Environments

In the industry, the traditional training of workers to use special equipment is normally carried out using a part or full real equipment. This could be afforded by the industry itself or specialized centers for training. But it brings many drawbacks like: the cost of equipment just for training is too high; machines are innovating and training equipment should change; new products or improvements of the production line which implies new training; outsourcing training with specialized centers, etc. Beside this kind of training there is also more specialized training like aviation or surgery where it is not always possible to use the real equipment and to check all the cases that the trainee could face.

Because of this, the help of computer solutions has been considered. They offer lower cost and more adaptability. The simulation of a working environment with computers is done by means of Virtual Reality (VR). In these applications we are able to build any kinds of scenarios, tools and equipment. However, a complete and detailed simulation of some scenarios appears to be very complex to develop, and moreover it is still difficult to produce truly convincing results.

Thus, to reduce the programming effort and also to simulate better the reality, Mixed Reality (MR) provides a good solution [110]. The principe of Mixed-Reality consists in superpositioning real images (pictures or video) inside of virtual world or vice versa. It can provide a complete real scene with virtual elements that help with the training process, as it is shown on the figures 7.1 achieved in the framework of the STAR European project.

These technologies are affordable and good enough to simulate working cases. They can show the proper way to play a role inside a context which is the primary goal of training. But usually these technologies are limited to keyboard or mouse interaction. In some cases other user interfaces are used, like large screens or touch screens. It is of course a big issue in a context like training. This is the reason that convinced us to study the integration of a two-handed device: The benefit of manipulating objects is to teach the user in a practical manner the proper way of performing tasks. For example, in assembly process: the user can manipulate virtual objects and position them directly.

In this section, we present a first a brief review of existing applications taking advantage of Mixed or Virtual Reality environment for manipulation or assembly training, with some of them providing force feedback. We remark these applications do not provide the ability to interact with real and virtual objects at the same time. It convince us to present a system that allows this kind of interaction. with the help of the Haptic Workstation$^{\text{TM}}$and of MHaptic. Then, by the mean of a simple feasibility study, we evaluate the benefits of such Haptic Mixed-Reality platform in the context of virtual assembly.

7.1.1 Related Works

When a haptic device is not available, a Mixed-Reality system presents the advantage over a purely virtual one that it can give the opportunity to use the real hands for interacting. For example, in [119], authors propose a training application where the user uses a see-through Head Mounted Display to see overlayed interesting information to help him to assemble furniture with his own hands. The drawback of such method is that it is limited to real objects manipulation. In [4], Azuma gives an overview on the recent advances in the field. In his article, haptic user interface are discussed as a new approach.

In VTT project [79], authors present a virtual technical trainer for milling machines, which is based on haptic devices. They use as prototypes three kinds of force feedback devices: the Phantom, a home-made 2DOF haptic device, and a pseudo-haptic technique. They present, in [34], an evaluation of these devices considering the efficiency criteria of the industry. We can also cite [74], which present a system assembly training in the field of aeronautic. Authors use a Phantom to simulate mounting/unmounting operation of different parts of an aircraft. In these exmaple, interaction is limited to virtual objects.

An example of interaction with real objects which moreover provides haptic feedback is in [86]. The authors use sensors to perceive the real environment, and transmit these sensors information to a 6-DOF haptic display with augmented force feedback. This is a truly *"augmented haptic"* system because the user is able to feel haptic textures of objects that he could not feel with is real hand (like the bumps of a sheet of paper).

An approach of real hands interaction with virtual objects is addressed by Walairacht et al. in [116]. They present a manipulation system of virtual objects where 4 fingers of each hand of the user are inside of a string-based haptic device allowing to feel the virtual objects. Moreover the video of the hands is overlaid on the virtual world to have a better visualization of the hand posture. Here again, the system is limited to virtual object manipulation.

In the next subsection, we present a two-handed manipulation system that provides the possibility to interact with real and virtual objects at the same time. The user is be able to use his both hands by the mean of the Haptic Workstation$^{\text{TM}}$.

7.1.2 System Architecture

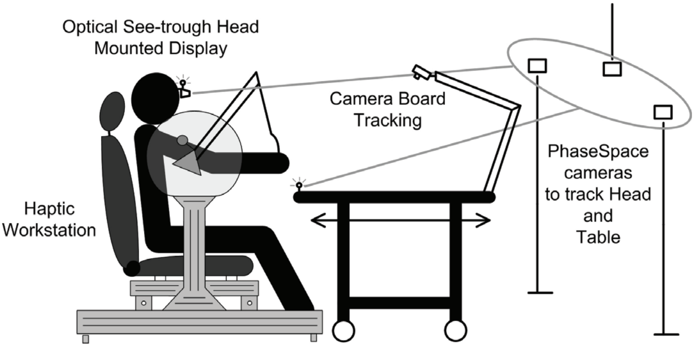

Optical See-trough Head Mounted Display
Camera Board Tracking
PhaseSpace cameras to track Head and Table
Haptic Workstation

Figure 7.2: General scheme of the four hardware modules of our application

As shown on figure 7.2 our application has been designed with many devices. First, the Haptic Workstation provides of course the haptic feedback. Then, it has also two tracking systems, and a see-through Head Mounted Display. Our assembly application, which is in fact a feasibility study, consists in building a Mixed-Reality table with a scale of $1/4$. It is constituted by a 55 cm long and 22 cm large piece of wood that contains also four holes where the feet are driven in. Four virtual objects represented by a $25cm$ long cylinder shape represents the feet. In the following subsections, we present the the Mixed-Reality integration –i.e. tracking system and HMD–. It has been of course programmed with MHaptic.

7.1.2.1 See-Through Head Mounted Display

In a mixed-reality system, virtual and real should be visually blended. Usually, two kinds of devices allow that: Video Head Mounted Display and see-through Head-Mounted Display (HMD).

Our implementation uses the *Sony Glasstron PLM-S700* see-through HMD. Advantage of such HMD in comparison with video HMDs is the quality of the real environment display: the reality is not “pixelized”. However, there is also drawbacks: they are usually semi transparent, and a virtual object could not completely occlude the reality. Moreover, the Glasstron HMD has tinted lenses (It could vary from opaque to a tint as standard sunglasses). Thus, the color of the real environment is altered. But it in a bright room, it does not really affect the user experience.

This HMD is calibrated using the SPAAM method [113]. It displays only the virtual feet because they are the only virtual objects of the scene (see figure 7.3).

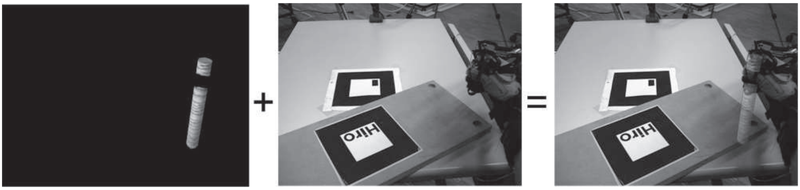


Figure 7.3: Photo taken from the user point of view, and augmented with what is displayed in the HMD

7.1.2.2 Tracking Device

Under mixed-reality conditions, real and virtual have to be well-aligned to avoid confusing the user. Moreover, with a haptic enhanced framework, real and virtual objects must collide each other, user should be able to interact with virtual objects as well as with real objects. This implies that we know the shape and the position of each objects of the system in realtime. This is not really a problem for the virtual objects, but, it is of course an unknown for real elements. As we have restricted our system to rigid objects, the shape of real objects could be statically stored. But the position and orientation values are dynamic, and have to be estimated for real objects during the simulation. In our feasibility study, three objects have to be tracked: the user's head (the HMD in fact), the board of the mixed-reality table, and the table where all the objects are putted (see photo and schema in figure 7.4 and 7.2).

We have used two different tracking methods. The first one could be considered as a software solution since it is based on the ARToolkit library with a common webcam. We choose to track the board with this method because it is truly wireless: nothing else than a piece of paper has to be attached to the board. The second one is the PhaseSpace Motion Capture system that we already used in section 4.2.2 to validate the CyberTrack calibration. This is the method that we choose to track the HMD and the work table (seen on figure 7.4 and represented on figure 7.2).

7.1.2.3 Assembly Training System

The hardware and software that we described in previous sections meet the requirements for creating a mixed-reality application. The real objects can interact with the virtual ones, because their position is known. The user is able to grasp a virtual foot. Then a simple haptic guidance in position system tries to move the user's hand in the location of the nearest board hole. This is achieved by applying a force vector to his hand whose direction is equal to the foot extremity/board's hole vector. The norm of the vector is proportional to the distance. When a virtual foot collides with one hole of the table and that the foot is perpendicular to the board, the force feedback response simulate the driving-in feeling.

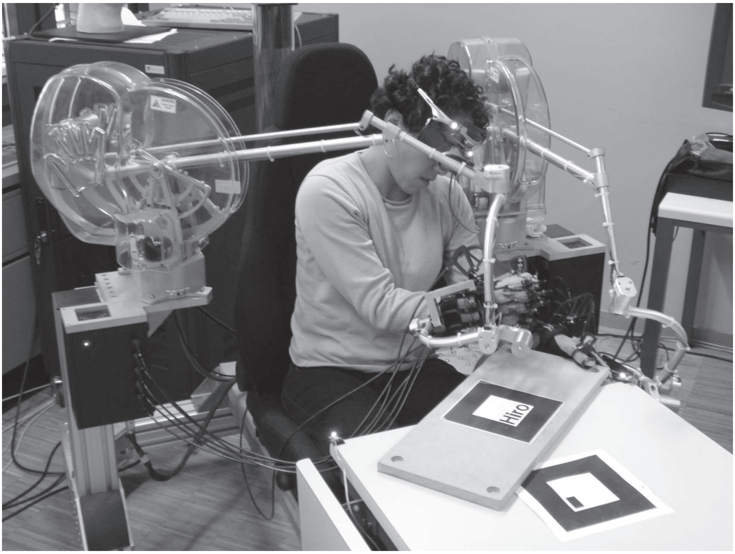


Figure 7.4: Photo of the devices used to build our Mixed-Reality system

7.1.3 Results and Evaluation of the System

In this section, we first present the testing protocol, and then we give a general evaluation of the complete system. Finally, we elaborate recommendations, based on our experience, to design a Mixed-Reality system that includes two-handed haptic interaction.

7.1.3.1 Experimentations

The described system integrates complex and heterogeneous VR devices that are not designed to work together. The calibration procedures of the devices could introduce errors, and the sum of these errors could lead to an unusable system. This subsection presents tests that will be useful to evaluate objectively these errors.

When dealing with mixed-reality and haptic applications, it is important to have an efficient mix between real and virtual. This is achieved by two components: the tracking of the real dynamic objects, and the projection of the virtual objects using the HMD. This lead to the first test which consists in measuring the difference between virtual and real environment: we ask to a user to grasp a virtual foot and to try to place it visually inside the hole of the table. Within perfect conditions, the system should detect that a foot is inside a hole and apply the “driving-in” force feedback. However two approximations have been done: first, the board position is evaluated by the tracking system; second, the virtual foot is displayed with the HMD and does not superpose perfectly on the reality. Thus, by measuring the distance between the virtual foot and the board's hole as they are stored in the system when they should be aligned, we approximate the addition of these two errors. We performed this test many times, moving the head and the board inside the workspace and we present the results on figure 7.5.

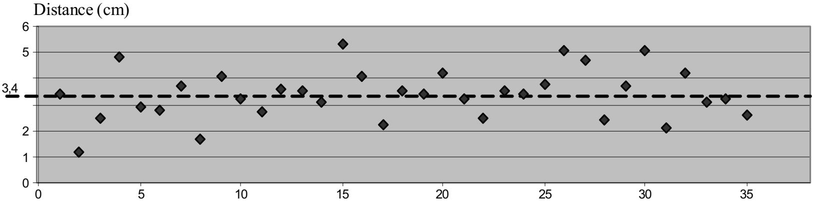


Figure 7.5: Distance between Real and Virtual environments measured by the first test (35 measures).

Second test quantifies how the user is perturbed by this difference: is he able to assemble the table under these conditions? In normal condition, the user sees only the real table board and the virtual feet. Thus, we compare the time taken to assemble this mixed-real table and the time taken to assemble a complete virtual table (without see-through). Finally, we have also done a test including the haptic guidance system: when the user grasps a virtual feet, he feels a force guiding his hand to the position where he can assemble the feet to the board. In this last situation, we can also evaluate if the user is perturbed of being guided to a place where visually, he is not supposed to assemble the table. To perform this test, we have ask to six persons to try the system. Usually, we ask to people that do not have a particular background in Haptics and Virtual Reality. However, in this case, we consider both the fact that the devices are complex, and that even if this system was applied to the industry the trainee should have a period of accommodation with the devices. Thus, we chose to ask to people knowing Virtual Reality devices (and especially the tracked HMD). Three “challenges” have been created:

1. To build the table in a completely virtual environment. Every real object is replaced by virtual one and simulated with MHaptic. The see-though HMD is set opaque.
2. To build the Mixed-Reality table.
3. To build the Mixed-Reality table, with the haptic guidance system.

The order is randomly sorted for each tester in order to cancel a kind of accommodation effect when we compute the mean time. We measure the time taken to perform these actions. Moreover, we gather oral feedback of the user after their test. We present the times in the table 7.1.

Test	1	2	3
Tester A	1m05	4m30	1m30
Tester B	0m55	2m00	1m25
Tester C	1m30	5m00 (Max)	1m50
Tester D	1m00	1m30	1m30
Tester E	0m45	2m10	1m15
Tester F	1m45	5m00 (Max)	2m10
Mean Time	1m10	3m02	1m37
Rank	1	3	2

Table 7.1: Times to build the Virtual and Mixed-Reality table by each user.

7.1.3.2 Evaluation and Recommandations

The previous subsection describes the testing protocol of our system. In this subsection, we extract results from it in order to finally elaborates recommandations when creating applications combining Mixed-Reality and Haptic Feedback.

The first test presents an important fact: despite all the calibration procedures, the matching difference between the real and virtual world is still high. The mean is around $3,4cm$, and the standard deviation is high ($0,95cm$): this is because errors are sometimes cumulated sometimes canceled. Moreover, with these results, we present only the difference norm: but we remarked that the "difference vectors" are in every directions of the space. Thus, it seems to be difficult to find a correction improving the matching using the hardware that we have. After more detailed investigation, the main errors in the calibration procedure are located at the display level. Using the optical see-through HMD calibrated with the SPAAM procedure, a displacement of this one on the face of the user during the manipulation is difficult to avoid. In [17], the authors have used a video-through HMD, device that avoid the difficult calibration of the HMD.

Second test shows that the assembly procedure is more easy when having only virtual objects, and that our mixed-reality system is not able to be as fast and efficient than an entirely virtual one. However, as mentioned in the introduction, it is sometimes impossible to have a completely virtual environment for many reasons (cost, complexity) and sometimes the goal of a training system is to teach using the real equipment itself. In these conditions, with a simple feasibility study, we have shown that it is difficult to manage haptic assembly with mixed-reality. This is mainly due to the visual sense that is not truly convincing. Hopefully, we have shown that some haptic techniques could help: the haptic feedback guidance, for example is very efficient in these conditions. The testers understand well that the virtual and real visual environment are not perfectly superposed, and that they will better apprehend the mixed-reality world with the help of the haptic guidance. Now, the main question is to evaluate how much the differences between virtual and real, visual and haptic, perturbs the learning curve of the trainee. According to the discussions with the testers, we believe that, in the assembly/manipulation context, the important point is the order of the actions/movements. In such case, haptic feedback and guidance is a good tool because it provides the enactive knowledge that the trainee should acquire.

Finally, we remark that these tests provide good indications on the way to build a haptic system under mixed-reality conditions. As it is explained in the previous paragraphs, the perfect visual matching is difficult to reach. Some studies on pseudo-haptic feedback have shown that the visual channel influences the haptic perception [73]. Thus, a realistic haptic feedback is not mandatory since it will be anyway perturbed by the haptic/visual misalignment. However, augmented haptic feedback like the haptic guidance mechanism provides a good solution to build an efficient system. This is the main result of this paper.

7.1.4 Conclusion

In this section, we have presented a system that allows training for manipulation and assembly tasks into a Mixed-Reality Environment. It allows to interact with real and virtual objects at the same time. With this feasibility study, we elaborate some recommendations when dealing with Mixed-Reality and two-handed haptic force feedback. Even with an efficient tracking system, mixed-reality techniques using optical see-through HMD are not precise enough to superpose correctly the virtual on the real world. The problem is that a small misalignment is acceptable when only the visual sense is stimulated. However, when combined with haptic force-feedback, the mixed-reality world will be much more difficult to apprehend, because of kind of ghost effects. The user feels something that he does not see, or the opposite. This is comparable to the mechanism of pseudo haptic techniques: the visual channel could “create” haptic feedback. Thus, trying to reproduce realistically an assembly situation in a mixed-reality with haptic feedback context will inevitably lead to a system that is difficult to use. But applying augmented haptic feedback to the user will improve the system usability.

7.2 Realistic two-handed Haptic Manipulation

In this section, we put the emphasis on MHaptic and its functionalities. First, we will show that is not limited to interaction with rigid object, but that it could be easily extended to soft or deformable objects. Then we present a application allowing the user to interact with the objects of a virtual house.

7.2.1 Soft Objects Manipulation

In section 5.2, about collision detection, we presented methods to optimize the computation of collisions between rigid objects. However, in reality, there are many objects that are not rigid. We can cite for example the clothes, the liquids, most of the organic objects. In the MHaptic framework, we are limited to rigid objects mainly because the *Haptic Scene Creator* and the *Haptic Scene* component do not allow to create or save other kind of objects. But it does not mean that we cannot create deformable objects: PhysX provides support of fluids and fabrics. We propose in this section to create a virtual carpet that can be stretched and torn. We present results in figure 7.6.

It is relatively easy to integrate this kind of object into MHaptic. Unfortunately, the PhysX library keep internal the results of the collision detection with such objects. Thus, it is impossible to have access to the data necessary to perform force feedback computation. It results in a proprioception that is not really well simulated.

7.2.2 Interaction with complex Virtual Environment

In section 6.2.4, we performed the augmentation of a complex visual scene representing a small house. In this section, we present the results obtained (in terms of performance) when manipulating this environment.

The PC running the simulation is a based on an Intel Quad-Core processor (2.4 GHz) and a NVIDIA 8800GTX graphics card. When loading the simulation, we remark that between 75% and 100% of the microprocessor is occupied, which is as expected. At the beginning during few second, the program lags. The reason is that every dynamic objects are in the “moving state”, and that most of them are colliding. Then, once they stop, the refresh rate is stable. The *Haptic Thread* is running around 900 Hz. The *Physics Engine* has been limited to 300 Hz, and the display is also refreshed around 300 Hz. The visual result is presented on figure 7.6.

When immersed into a large scale Virtual Environment, the user is not able to reach every object. As he is seated in the Haptic Workstation$^{\text{TM}}$, it seems difficult to propose a realistic displacement method. Thus, a metaphor should be used to let him move aground the Virtual Environment. In [41], authors presented an evaluation of 3 different techniques to achieve the interaction with objects that appear bigger than the workspace of the haptic device. We used a method similar to the “bubble” technique described in [40]. When the user moves his arms near the workspace limits, he enters into an area that displace the virtual camera: arms to the front moves the camera forward, arms to one side turns the camera. In addition, force feedback is applied to tell the user that he is entering to this area. This convenient method is intuitive, and the displacement is really controllable. However, we remark that a calibration has to be done to adapt the method to the user’s morphology. Indeed, the far limit of the workspace is not at the same position for everyone.

The force feedback is efficient with most of the static objects. We were surprised that even the thin tables produce a convincing resistive force due to the fact that the hand does not go through. The main limitation comes from objets augmented with pMaps. Even the static ones produce unexpected behaviors. It also occurs occasionally with dynamic objects that use convex meshes. Despite these issues, the feeling of being there (presence) seems increased, mainly because of the 3D interaction capabilities of the two-handed haptic device.

In this chapter, we presented applications with realistic interaction paradigms. In the first section, we proposed a study using an assembling training system under Mixed-Reality conditions. We created a feasibility study that allows to perform a test on the efficiency with different kinds of realistic interfaces. We presented interesting results concerning precision, and proposed useful recommendations about the design of such systems. Then, in the second section, we showed some Virtual Reality applications made with MHaptic. The common point between these applications is that they are **intuitive** thanks to the two-handed haptic device.

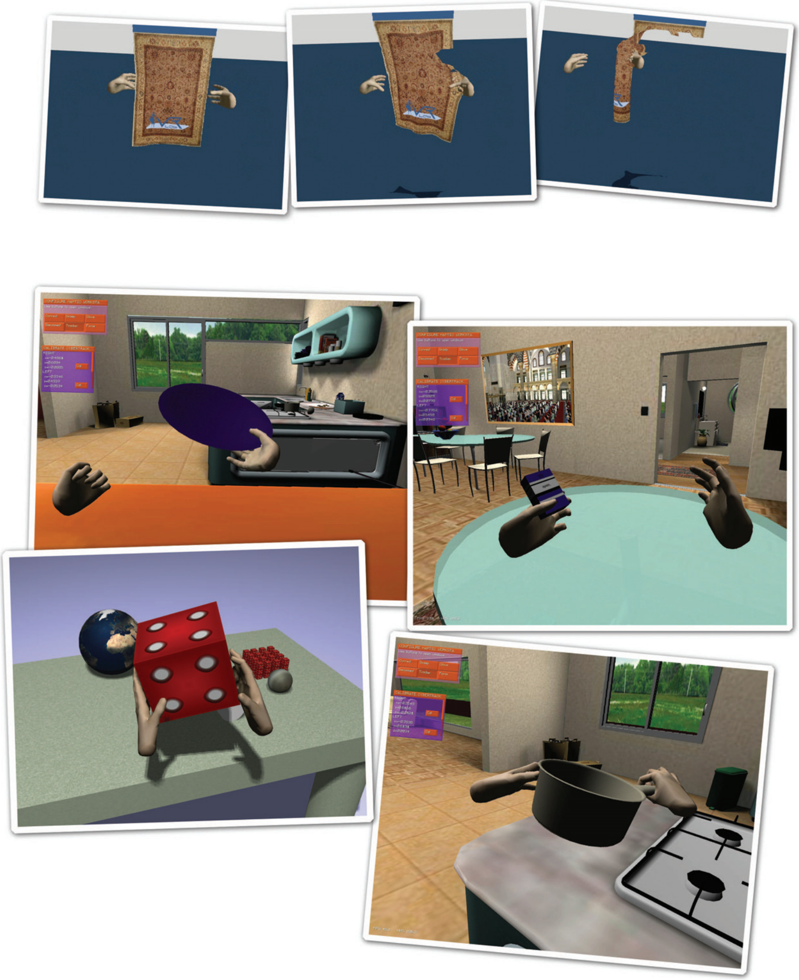


Figure 7.6: Manipulation and Interaction with a haptic house.

Chapter 8

Unrealistic Human-Computer Interactions

We showed in previous chapter that some applications, like Virtual Training, take advantage of realistic two-handed interactions. For an efficient training, we can distinguish the theory learning from the gain of experience. A two-handed interactive system can propose real life situations that increases the experience of the trainee. Some other applications do not simulate a real life interaction. They propose a paradigm based on metaphors, which is often the case in computer applications. The main disadvantage of this kind of interaction, is that it could require some kind of training and practice. But, when the experience is acquired, it could become really easy to perform complex tasks. In this chapter, we want to study these interaction paradigms, in order to see how a two-handed device could increase the learning speed without compromising the efficiency.

As mentioned in subsection 2.2.1, teleoperation is historically the first field that took advantage of Haptics [61]. This is one of the reason that convinces us to evaluate this novel platform in this historical context. We propose to create a teleoperation platform that uses Virtual Reality techniques and a two-handed haptic device. We perform two experiments with two different categories of teleoperated robot: one for driving a small robot, and the other one with the remote control of a blimp. One may ask: “why two experiments with two robots ?”. In fact, we think that a *specific interface for a specific robot* is needed to maximize the efficiency in terms of remote control. And we are quite sure that a generic device, even with a high number of degrees of freedom, cannot perform better than a specific interface. But, a generic device has the advantage to be easily **reconfigurable** to be adapted to many robots. This is the main characteristic that needs to be evaluated. The sections 8.1 and 8.2 present these two experiments.

8.1 Interaction Paradigm for Teleoperation

In this section we present the evaluation of different kinds of interfaces to remote control a small robot. We deal first with the technical description of the overall system, then we present our evaluation protocol, which is based on a robot race, and we finally discuss the results and conclude.

8.1.1 A Teleoperation System

This subsection deals with the implementation of our system. It uses the Haptic Workstation and it is based on the concept of mediators. Mediators are virtual objects with haptic feedback that act as intermediaries between the user and a complex environment. We introduced in [77] and demonstrated its application within an interactive Virtual Environment. In [57], we took the next step and presented the implementation of a mediator interface to drive a real mobile robot. The same robot is used in this study. The system architecture can be divided into two main parts:

- Controlled world: a mobile robot made up with the Lego® Mindstorms toolkit controlled by a laptop.
- Mediator world: a Virtual Environment with haptic feedback provided by a two-handed haptic device.

Both systems are connected to the Internet network and communicate between each other using the TCP/IP protocol.

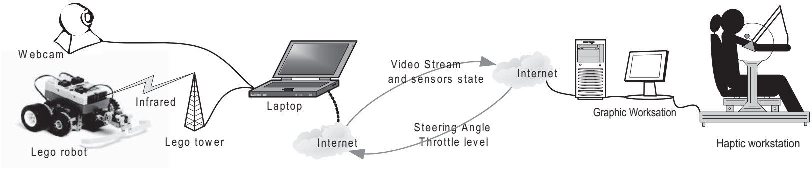


Figure 8.1: Controlled World on the left, Mediator World on the right.

The controlled world elements are illustrated on figure 8.1. The robot is a toy tank equipped with a collision detection sensor on the front-side and a web-cam. It is built using the Lego Mindstorms Robotics Invention System 2.0® kit. The robot is controlled by the RCX (yellow brick) which contains a built-in microprocessor. It can execute autonomous behaviors coded on a PC and loaded through the brick's IR interface, but we chose to bypass this functionality and use the RCX as a gateway for direct controlling the motors/sensors through a laptop. The motors can turn in both directions with 8 different speeds. By setting two different speeds on the engines, the robot can turn. On the front, there is a bumper with two boolean contact sensors which allow for collision detection on the left or right sides of the robot. The laptop runs a program to evaluate the boolean position of the sensors and turn on the motors to move the robot. Communication between the laptop and RCX is done trough the infrared port of the RCX and the Lego USB Tower connected to the laptop. The video stream is acquired with a Logitech Quickcam P4000$^{\text{TM}}$ located on top of the robot and connected via USB to the laptop.

The mediator world shown on figure 8.1 is composed by a 3D graphics Workstation (Windows PC) and our Haptic Workstation$^{\text{TM}}$. The Workstation renders the Virtual Environment for the user seated in it. To drive the robot, the pilot has different types of interfaces, described in the next subsection.

Three main kinds of data streams are exchanged between both worlds, they are also illustrated on figure 8.1:

- Video stream coming from robot to the mediator world.
- Messages coding the position of the virtual cockpit elements.
- Messages coding the state of the contact sensors of the robot.

In the next subsection we proceed with the presentation of the teleoperation scenario we have implemented and describe the test protocol we have followed.

8.1.2 Teleoperation scenario: a robot “grand-prix”

The teleoperation scenario is a car race around obstacles with a few bends as illustrated in figure 8.2. The goal is to complete the circuit as fast as possible. The very limited speed of the robot and the ease of the circuit guarantees that the driver’s expertise will not be determinant in the time required to complete a lap. This is important when evaluating the efficiency of the interface. Before having test users try the interface, we measured the optimal time required to complete the circuit, by driving the robot directly from the controller laptop, using the keyboard and watching the robot directly (see Figure 8.2). The optimal time is 1m30s.

Four different types of mediator interfaces, which will be defined in next subsection, will be tested by each test user. They have to evaluate the *efficiency* and *intuitiveness* of each variation. By efficiency we mean the capacity of the interface to let the user accomplish the workload satisfactorily. The workload in this case consists on: first, finishing the race; second, avoiding all the obstacles; and third, doing it as fast as possible. *Efficiency* can be objectively measured in terms of the time taken to finish the lap and the number of obstacles touched. *Intuitiveness* is more subjective. It refers to the ease of learn and use the interface. We measure it by means of a questionnaire and direct observations of the user’s behavior when using each interface. The evaluation criteria and analysis methodology will be detailed later in this subsection.

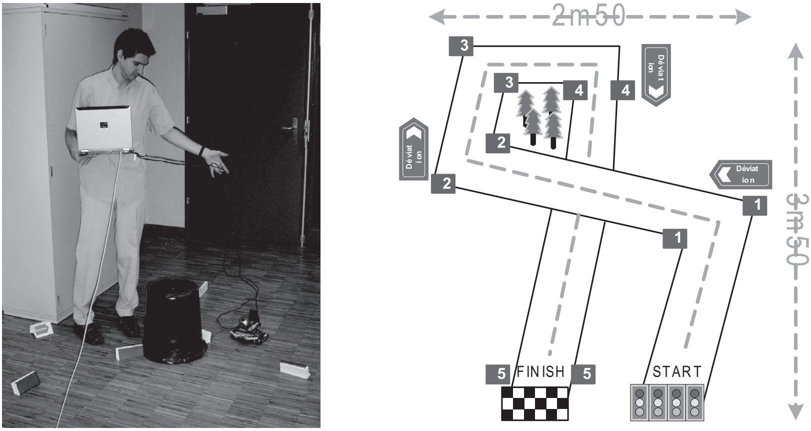


Figure 8.2: The robot Grand-Prix

A test with several persons is the only way to validate our search for the best interface. We choose a group of persons who are not working with haptic technologies to perform the test. Users are from 25 to 40 years old, three men and one women, all of them with a Computer Science background.

We describe now the specific protocol used to execute the tests.

8.1.2.1 The protocol

We introduced some randomness in the testing procedure. Each user must test four different interfaces, but the order in which each interface is tried by each user is random. This was done to minimize the effect that after some trials, people can get used to driving the robot and finish the lap successfully even with an inefficient interface.

Before the tests, the driver is allowed to do a lap with a remote-control and direct view on the robot to study how it turns and moves. The remote-control is a PC that displays the visual feedback of the robot's webcam and controls the steering wheel and the throttle with the keyboard. It gives the opportunity to the user to perform a mapping between the exact position of the robot and the visual feedback of the webcam. This also gives some points of reference which can be helpful to decrease the difference between the first and the last test performed by the driver. then, the user is placed in another room, and do not have a direct view on the robot. The evaluation is based on a set of well defined and easily measurable parameters defined as follows.

8.1.2.2 Evaluation parameters and analysis

There are four race results corresponding to each interface for each person. There are two evaluation parameters used to benchmark the interfaces:

- Global lap time spent on each interface.
- Ranking each interface on a per-driver basis.

The first parameter is obtained by adding the time spent by each driver to finish the race using a given interface. Then, the best interface will be the one with the smallest time. The second parameter is calculated by ranking the interface according to the performance of each person. On a per-driver basis, the best interface is the one that allowed finishing the faster lap. The best interface will be the one that was best ranked by all users.

This benchmark does not take into account subjective criteria required to evaluate the intuitiveness. Thus, we ask the testers to answer a small questionnaire and we complemented the analysis making an evaluation of the overall performance of each interface.

8.1.2.3 Measuring intuitiveness

The questionnaire used to evaluate the driver's impressions about an interface was composed by 3 questions:

- Is the interface easy to learn?
- Do you think this interface is efficient to drive the robot?
- Do you have any remarks about this interface?

We asked these questions to the users after they tested each interface. The objective was to identify contradictions between performance to complete a lap and user perceptions (interface intuitiveness).

8.1.2.4 Overall evaluation

The test and responses to the questionnaire were complemented with an overall performance evaluation of the efficiency (the capacity of the interface to help the users complete the workload). The overall evaluation was done by giving the interface a mark according the rating scale for teleoperation systems shown in figure 8.3.

This method has been proposed in [24] to evaluate military cockpits. It was also later applied in [43] in which the authors presented an approach to evaluate two interfaces for the control of robotic surgical assistants. We adapted this evaluation to our own task:

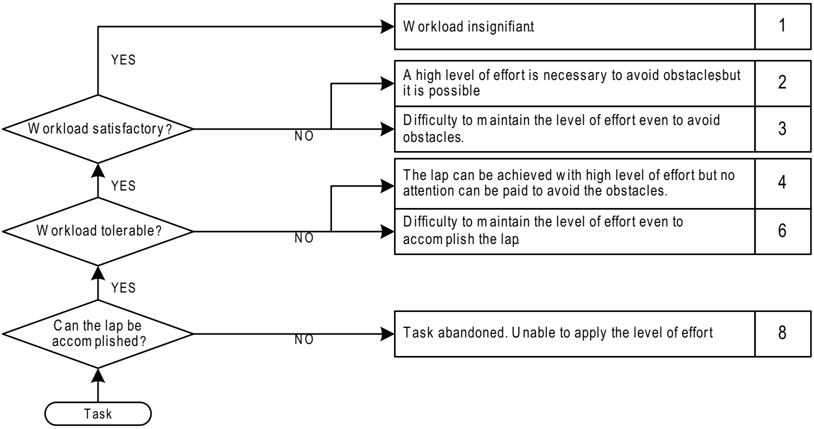

Workload insignifiant
1
YES
A high level of effort is necessary to avoid obstacles, but it is possible
2
Workload satisfactory?
NO
Difficulty to maintain the level of effort even to avoid obstacles.
3
The lap can be achieved with high level of effort but no attention can be paid to avoid the obstacles.
4
YES
Workload tolerable?
NO
Difficulty to maintain the level of effort even to accomplish the lap.
6
YES
Can the lap be accomplished?
NO
Task abandoned. Unable to apply the level of effort
8
Task

Figure 8.3: Rating scale for teleoperation systems.

measuring the efficiency of a teleoperation interface for driving a robot on a circuit (primary task) while avoiding obstacles (secondary task). This rating scale allowed us to have a unique mark characterizing each interface.

In next subsection, we discuss the alternative mediator interfaces we have implemented, and give results of the driving tests and responses to the questionnaires.

8.1.3 Alternative mediator interfaces

To control the robot, four alternative mediator interfaces have been designed. They are shown on figure 8.4 (second and third interface are visually identical). The design pattern consists in going from physical/realistic cockpits up to gesture-based interfaces. The evolution is quite natural, the first interface is based on real car cockpits whereas the last one takes advantage of the Haptic Workstation$^{\text{TM}}$as a system designed to acquire and drive (through force-feedback) the arm gestures performed by the user.

All interfaces have a common visual element: a virtual screen that displays the video stream sent by the robot webcam. This element is essential to know the location of the robot in the remote scenario. Additionally, all interfaces have a common haptic behavior: in case of collision between the robot and an obstacle, a signal is sent to the interface and controls are blocked to prevent the user from keep moving toward the obstacle. Implementation differences will be described in the next subsections.

8.1.3.1 First Approach: Having virtual elements which look like reality

The first approach tends to reproduce a standard vehicle cockpits as shown in figure 8.4. The steering wheel and the throttle are universal interfaces used to control a car, so it seems logic to use a virtual cockpit which looks like a real one. The mediator interface is thus composed by the following parts:

- A haptic and visual virtual steering wheel.
- A haptic and visual virtual throttle.

The haptic shapes of the steering wheel and the throttle are exactly the same than the corresponding visual shapes. When a collision is detected by the contact sensors of the Lego® robot, the virtual steering wheel shakes and the throttle is blocked a while. This behavior is the same for all three interfaces with these controls.

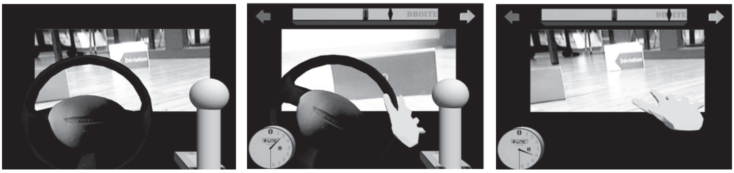
The time taken by each driver to perform a lap is shown on table 8.1, and the per-driver-rank of of the first interface. The last line presents the number of gates which have been missed or touched by the driver:

People	A	B	C	D	E
Time	3m30	10m00	4m05	5m20	5m10
Rank	3^{rd}	3^{rd}	4^{th}	2^{nd}	4^{th}
Obstacle	1	5	1	1	1

Table 8.1: Results of simple interface test.

In this test, the driver B has reached the time limit: he has driven during 10 minutes without passing trough the 5 gates. Thus we decided to set his time to the maximum time in order to not penalize too much the interface in the global time ranking.

After discussing with the testers, we found the first advantage of this interface is obvious. Nobody has asked for the use of the controls: it is very intuitive that the steering wheel



First Interface

Second and third Interface

Gesture Interface

Figure 8.4: Alternative mediator interfaces.

controls the direction and the throttle sets the speed of the robot. If the user has nobody to help him, he can still understand easily the use of the interface.

Besides this advantage, this interface is not very convincing. First of all, drivers criticize the visual feedback. Everybody has touched at least one gate. Frequently, obstacles were not visible on the screen because the camera was placed on the front of the robot, and the view angle was not large enough. Moreover, there is no speed or direction perception. These two points often make the driver think he is too far and he stops running before passing trough the gate.

In order to improve the perception of speed and direction we decided to add complementary visual feedback to give a better idea of the robot motion to the driver; in an analogous way as the HMD used by jet pilots which provides them useful information.

8.1.3.2 Second Approach: Adding visual feedback to enhance control

The drivers need more information about speed and yaw of the robot. Thus we choose to add two visual elements (see figure 8.4):

- A visual speedometer which displays the current speed of the robot.
- Two indicators flashing when the user turns.

The table 8.2 presents the results obtained for the second interface.

People	A	B	C	D	E
Time	3m45	10m00	3m40	6m45	4m00
Rank	4^{th}	3^{rd}	3^{rd}	3^{rd}	3^{rd}
Obstacle	1	5	0	1	0

Table 8.2: Results of added visual feedback interface test.

This second interface has very similar results to the first one. Addition of drivers times is 28m05s for the first one and 28m10s for the second. Means of rank are equal and obstacle collision differs from two. The only conclusion we could draw by considering the results of these tests is that the additional visual feedback does not provide sufficient helpful information.

By discussing with the drivers, we discovered that they did not really look at the speedometer because they gave priority to the task of controlling the robot. These tasks are so hard that they considered the collision avoiding as a secondary problem they did not have time to treat. A new question appears: why is it so hard to control the robot? The steering wheel is hard to turn because the Haptic Workstation is not fully actuated to simulate properly this mechanism. This implies that the driver concentrates more on grasping the steering wheel than on driving. We try in the third interface to simplify the use of the cockpits elements. We add them a “return to zero” functionality: when the driver releases a control, this one comes back to its initial position. By this manner, the driver spares on the one hand the movement necessary to reset it. On the other hand the effort to aim the center (the initial position) of the control is spared as well. The third interface takes advantage of this constatation.

8.1.3.3 Third Approach: Adding assisted-direction to interface elements

The visual aspect of the third interface is exactly the same as the second). It differs from the precedent by enabling the “return to zero” functionality.

Results for the third test are presented on table 8.3.

People	A	B	C	D	E
Time	2m50	2m30	3m10	5m25	3m40
Rank	2^{nd}	2^{nd}	2^{nd}	3^{rd}	2^{nd}
Obstacle	0	1	1	1	0

Table 8.3: Results of assisted direction interface test.

Except for the user D who tested this interface at the first, every drivers have found that this interface was better than the both precedents. Total time spent on it was 17m35s which is a significant decrease in comparison with the first and second one.

Responses to the questionnaire provided by the drivers permit us to understand that the “return to zero” functionality is very helpful. It is not intuitive when the user sees the interface for the first time, but after having touched a cockpit element, he/she understands quite immediately its behavior without surprise.

Nevertheless the lap times are the double of the perfect lap time. In fact, the drivers do often unintentional changes of orientation because the Lego® robot does not have a smooth behavior while turning. When it happens the time taken to recover the right direction could be significant, and increase even more if the driver tries to turn faster to spare time.

Some people used only one hand to manipulate both controls, because they found too hard to use them at the same time. This problem comes from the lack of feeling of the control. Proprioception is not enough to feel the control. Tactile feedback is also important for this task. Users counter this lack of haptic feedback by watching the hands. Of course, it appears to be hard to watch two virtual hands and the webcam feedback at the same time.

Currently, hands interact with the controls (steering wheel, throttle) and then a mapping between controls position and robot engines is done. In this process, the controls are an additional intermediary component which could be eliminated in favor of a direct mapping between the hands position and the robot engines as presented on figure 8.5. This is how we came up with the forth mediator, a gesture-based interface.

8.1.3.4 Fourth Approach: Gestures-based interface

In this interface, we remove the virtual controls and the left hand (see figure 8.4), but we choose to let the indicators and the speedometer because they don't complicate the visual interface and drivers can use them from time to time.

Obviously both haptic shapes (used to generate the force-feedback) corresponding to the controls are removed, and a field of force constraining the right hand at a comfortable position is introduced. The user can still move his/her hand everywhere, but the force becomes stronger proportionally to the distance between his/her hand and the neutral position.

Figure 8.4 presents the results of each driver with the gesture interface.

People	A	B	C	D	E
Time	2m00	2m00	1m50	3m30	3m20
Rank	1^{st}	1^{st}	1^{st}	1^{st}	1^{st}
Obstacle	0	1	1	0	0

Table 8.4: Results of gesture interface.

The rank convinced us totally: all users did their best lap with this interface, even for user E who started the test with it (and did not have the same level of familiarity with the system). Best times are not so far from the optimal lap time (1m30s). The difference may come from the view angle of the webcam, which is much more limited compared to the direct visual driving.

The unique disadvantage we found in this method is that the user does not really understand how to drive the robot just by looking at the visual interface (less intuitive at first sight). However, the reflex of moving the hand to interact with something comes quickly and then, after a few tests, the user learns to drive.

The control on the robot is precise. Users can change direction in a simple movement, and go forward and backward by the same manner. When a collision is detected with

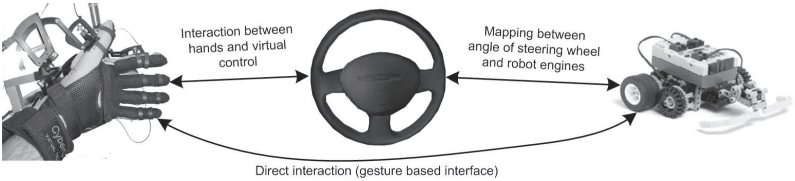

Interaction between hands and virtual control
Mapping between angle of steering wheel and robot engines
Direct interaction (gesture based interface)

Figure 8.5: Mapping between hands, virtual controls and robot engines and short-cut used to create a gesture-based interface.

a gate or a wall, the haptic response is more intuitive than shaking the controls, and the user understands better what happens. Moreover, one really feels a “wall” preventing any further motion of the hand towards the obstacle. In contrast, with the virtual controls, users often thought the blocked control was either a bug in the system or a lack in their driving skills.

In the next subsection we present the overall evaluation of results, and we state our conclusions about teleoperation of robots with a Haptic Workstation$^{\mathrm{TM}}$.

8.1.4 Discussion of results

Several lessons have been learned from the tests and the questionnaire. Figure 8.6 sums up all tests results, and confirms that our intuition about the gesture interface was well founded: it revealed to be the most efficient interface, but perhaps not the most intuitive one.

The overall evaluation obtained using the method described in Figure 8.3 confirmed the ranking obtained with the other benchmark (time to finish the lap, per-driver ranking): the most efficient interface, the one that minimized the effort to accomplish the workload was the gestures-based interface. In second place we have the one with “assisted-direction” and the last place is shared by the two first approaches. We believe we were able to avoid influence from the driver’s skills when evaluating the interfaces, since even with the worst performer, the gestures-based interface was the best evaluated.

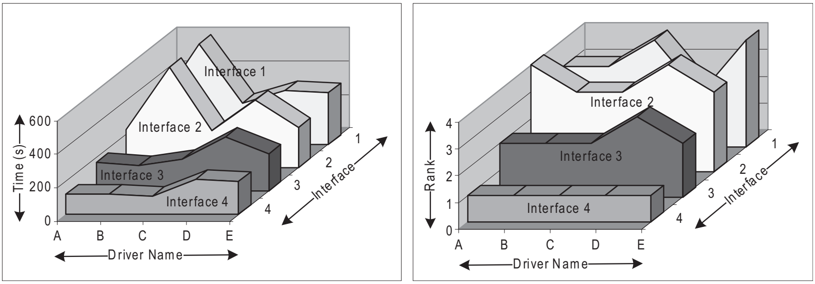


Figure 8.6: Overall results of the driving tests.

The gesture interface eliminates the interaction between the hands and virtual controls and for the moment seems to be the best approach. As long as hardware does not allow enough haptic feedback on both hands and arms, it will be difficult to have a good perception of grasping and manipulating an object like a steering wheel.

We can conclude that the efficiency of the teleoperation interface depends significantly on minimizing the actions of grasping objects to trigger functionalities (such as the mapping between the steering wheel and the robot engines).

Based on the presented tests we draw the following general conclusions about the factors that affect the efficiency and intuitiveness of an interface for teleoperation:

An efficient interface for direct teleoperation must have rich visual feedback in the form of passive controls such as speedometers, direction indicators and so on. Such visual aids were appreciated by users once they were released from the burden of manipulating the virtual steering wheel and throttle.

Force feedback shall be exploited not as a way to simulate tangible objects (interfaces resembling reality) but to drive the users gestures (gestures-based interface).

As reported by the users, the gestures-based interface was efficient because it did not required precise manipulations. It reduced the amount of concentration required to drive. The user could direct her attention to the rest of the visuals and use them to improve the driving.

We can consider the gesture interface as an adaptive interface that does not impose the driver to perform a well defined gesture but only to reproduce a general intuitive pattern (moving the hand forwards, backwards,...).

Virtual interfaces that resemble reality were the most intuitive ones, in the sense that users knew immediately how they worked (previous real-world experience). Nevertheless, the available hardware made them less efficient due to the problems with the grasping mechanism explained before.

It is finally important to notice that the observations and assumptions presented here can be strongly dependent on the hardware used and the teleoperated robot. Perhaps an ad-hoc designed hardware (the Haptic Workstation was conceived as a multi-purpose equipment) could give better results in terms of grasping and manipulation. A more responsive robot equipped with some degree of autonomy could assist the user and compensate for the drawbacks due to unprecise grasping. Nevertheless, the adaptivity of the gestures-based interface is a very important element to keep in any type of interaction mechanism.

8.2 Teleoperation of an unmanned aerial Vehicle

In the previous subsection, we conclude that a gesture-based interaction paradigm is preferable to teleoperate a robot with a two-handed generic device. We show also that the performance of the interface is near optimal in this specific case. In this section, we present the extension of this system to the teleoperation of a blimp.

8.2.1 Another teleoperated vehicle: the R/C Blimp

Our blimp, as shown on figure 8.7, is a low-cost Unmanned Aerial Vehicle (UAV) that we use in our teleoperation research. The *R/C Blimp* is composed by a $6,75m$ long and $2,20m$ diameter envelope that is filled with $11m^3$ of Helium gas (He). The total weight

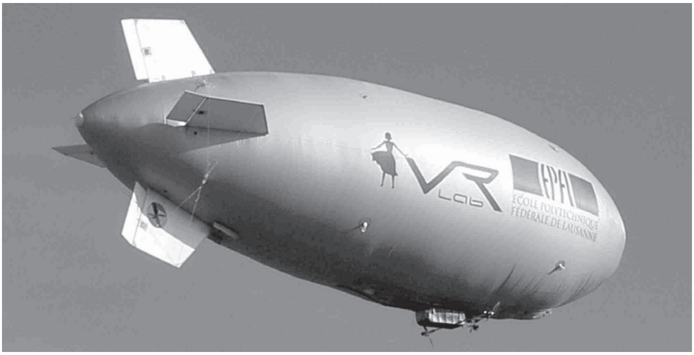


Figure 8.7: Photo of the R/C blimp.

including its standard flight equipment is $9kg$, so there is around $2kg$ ofmaximum payload for the cameras and the video transmission system. Below, there is a gondola containing the electronics part, the power supply ($6600mAh$ allowing $1h$ at half-speed) and supporting the two electric motors. Each have $1,5kg$ of power, allowing the blimp to fly at $35km/h$ when there is no wind. The range of thetransmission of the radio controller is $1.5km$, but it can be extended with repeaters.

Figure 8.8 shows the five actuators controlling this *R/C Blimp*. Aerodynamic stabilizers are used only when the blimp has reached a certain speed (approximatively $8km/h$). In previous subsection, we have seen that the agility is essential when flying in urban environments. Thus, there is one extra-helix, located on the bottom stabilizer, that can run on both directions. It is used to turn left or right when the Blimp is flies at low speed. The pilot can also change the plane of the main helixes, in order to go up or down at low speed. Finally, there is an actuator for the speed which controls the rounds per minute of the helixes.

Part of the $2kg$ payload is for carrying two video-cameras and their transmission systems. The first camera can be controlled by the pilot using head movements. The orientation system must be faster than the head, and the field of view has to be large enough

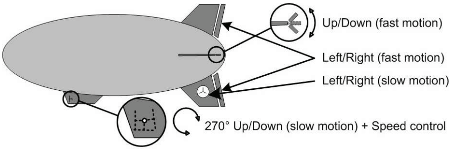


Figure 8.8: Actuators of the blimp: aerodynamic stabilizers, left/right motor, motors for changing helixes plane and speed.

to permit a good control of the *R/C Blimp*. Thus, we chose a small (less than $50g$) wide-angle CCD-camera. The other camera is used by the *Surveillance Control Room*. Thus it must have a good quality, allow for recording and zooming: we have chosen a Panasonic mini-DV TriCCD camera ($1.2kg$). We use the analogical output of these cameras with two systems of video transmission. On the ground, we receive two PAL/B signals that are digitalized. Both are connected to a workstation located on the network that broadcasts the streams.

Finally, the actuators are controlled by a $40MHz$ 7-channels Futaba FX-18 radio controller. On board, the reception system is completely duplicated to prevent failures. Each of the two airborne cameras can be oriented along the pitch and the yaw angle (two channels), and the Panasonic camera can also zoom (one channel). The wide-angle camera that is used by the pilot is also connected to the Futaba radio, so the pilot has to control seven channels. We have used an USB SC-8000 Servo Controller that allows a PC to control a radio.

8.2.2 The Virtual Cockpit

In this subsection, we will describe how we have implemented the cockpit using Virtual Reality devices. The *R/C blimp* is not so easy to pilot, even with the remote controller, which is usually the classic device for this purpose. Moreover, the *Virtual Cockpit* is in an isolated room without any direct-view of the *R/C Blimp*. Therefore the interface must be **precise** (for a fine control) and **intuitive** (to avoid manipulation errors). In the first part, we deal with the visual rendering of the cockpit, after we describe the haptics part of the interface. Both of them tend to achieve the two goals mentioned above.

The visual part of the blimp is rendered to the pilot via a Proview^{TM} XL50 Head-Mounted Display (HMD). This HMD gives a 1024×768 resolution on both eyes, with an horizontal field of view of $40°$. In order to have a virtual camera that moves according to the head movements, we have used an orientation tracker, the InertiaCube3, provided by Intersense. This tracker is very precise and has a refresh rate of $180Hz$. On software side, we are using MVISIO to render the Virtual Cockpit, which consists in a stereo 3D model of a blimp cockpit and a polygon where we map the realtime video stream. Figure 8.9 shows the representation of the blimp inside the virtual environment.

In accordance with the results obtained in 8.1, we use a gesture-based interface: the right hand is used to control the aerodynamic stabilizer (by moving the hand forward/backward and left/right), whereas the left hand is used to control the engine power. However, when there is no power, a left/right hand movement does not move the stabilizers but controls the rear engine allowing the pilot to do faster turnabout. The force feedback constraints the user hands to the neutral position, i.e. no power and straight ahead. When the user wants to move a hand, the force feedback intensity increases. The pilot can thus feel how fast he is going by evaluating the force applied on his hands.

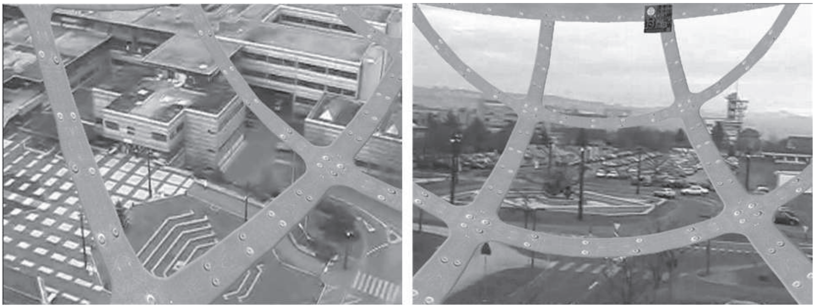


Figure 8.9: The Blimp's Virtual Cockpit.

8.2.3 Results

The goals were to have a precise and intuitive interface. We have also observed that the Haptic Workstation^{TM} is precise enough to offer a fine control on the blimp's actuators. We successfully made a 5 minutes teleoperated flight with a takeoff, and passage over buildings. We did not performed a landing, not because of the lack of control, but because it was too risky. In conclusion, the gesture-based control seems to be a promising technique.

The use of a responsive head orientation tracker to drive the video camera is also an efficient method. It allows indeed to have mixed reality video stream coming from the *R/C Blimp* that is well positioned into the virtual 3D cockpit. This is also an intuitive way to move such a camera because it reproduces exactly what happens in the reality. Thus, it increases the believability of the *Virtual Cockpit* by giving to the pilot the impression that he is inside a real Blimp.

In this chapter, we have successfully implemented two efficient Virtual Reality interfaces for teleoperation. The use of a two-handed haptic device for simulating the controls is probably not as powerful than two dedicated interfaces. However, it provides the ability to control two different devices. We can also imagine that two or more devices can be teleoperated by the same pilot. Indeed a Virtual Cockpit is easily reconfigurable. Thus a pilot can quickly switch from one to another robot, while autopilots control for a while the rest of the vehicles.

We group these experiments in the category of *"unrealistic interaction paradigm"*, because we demonstrate that even a state of the art two-handed haptic device is not able to render efficiently simple tools like a steering wheel. The lack of feedback and the underactuation is the cause of efficiency problems. The creation of other paradigms solves partially the efficiency issue to the detriment of intuitiveness. In [92], we also proposed an interface based on the Haptic Workstation^{TM} to model 3D shapes using metaballs. 3D modeling is typically a field where there are a lot of metaphors in the interaction paradigms. Our main idea was to find out if we can provide an efficient paradigm that is simply based on hands and gestures. Results are mitigate: the provided method requires too much training and manipulation is thus not as simple as expected. This category shows the limits of a two-handed haptic device like the Haptic Workstation^{TM}.

Part IV

Synthesis

Chapter 9

Conclusion

WE BELIEVE THAT two-handed Haptic feedback is a wide topic which is still under exploration. When Guiard analyzed human bimanual action, he proved that the vast majority of human manual acts involve two hands acting in complementary roles [55]. However, in Haptics, the majority of undergone experiments involve only one hand, or even one finger. In this thesis, we made preliminary studies on two-handed haptic feedback focusing, on the one hand, on the *realistic rendering for manipulation*, and on the other hand, on its *applications*.

9.1 Summary

In this thesis, we first have presented, in chapter 2, a review of the existing haptic devices, which can be separated in two main groups: tactile and kinesthetic devices. We also proposed a review of the Virtual Reality applications that take advantage of the force feedback and 3D interaction provided by kinesthetic devices. At the end of the chapter, we presented works related to whole-hand and bimanual interaction, from a cognitive and computer sciences point of view. Our main conclusion was that there is a clear lack of two-handed whole-hand haptic systems.

In chapter 3, we studied the needs to interact in a natural manner with a Virtual Environment. We demonstrated the need to track the two hands –i.e. their posture and position– and to provide force feedback on them. Mechanical haptic devices with such properties are definitively not common. In fact, the only commercially available one, is the Immersion® Haptic Workstation$^{\text{TM}}$. The others are research products. The end of the chapter describes this device in detail.

The second part of this thesis focused on the software control of the Haptic Workstation. In chapter 4, we presented an optimized method to access to the device in order to retrieve useful data as fast as possible. However, it is impossible to use the data in this form because it is not calibrated. For this reason, we presented also fast registration methods suiting the needs of realistic haptic manipulation. Finally, in this chapter, we proposed a software method to improve the user's comfort while interacting with Virtual Environments. This strong base eases the creation of the remainder of the haptic rendering software.

In chapter 5, we proposed and described MHaptic. MHaptic is a multithreaded library that handles the haptic rendering and the animation of virtual objects. First, we presented a study on the general structure of the library. We described the modules that are working together in different threads, and the ways to achieve an efficient synchronization between them, in order to maintain high refresh rates and data integrity. We then presented the optimized techniques used in the collision detection and realistic animation modules. Finally, we proposed our haptic hand model which is based on a mass-spring system managed by the physics engine. The model eases force feedback computation and allows generic manipulation of virtual objects.

In chapter 6, we proposed techniques allowing a user to interact with any Virtual Environment. The fact is that existing Virtual Environment models usually do not contain properties needed to compute force feedback. Starting from this observation, we presented a study on the parametrization of the mass-spring hand model and of the virtual objects in order to ease the grasping and to improve user's feeling. Then, we proposed and described a software, the *Haptic Scene Creator* whose role is to edit an existing visual Virtual Environment in order to add these missing properties.

In the third part of this thesis, we presented experiments in some applications that we believe they can take advantage of two-handed haptic rendering. We separated these experiments into two groups. The ones that feature a realistic interaction paradigm, and the others. In chapter 7, we performed a study on two-handed haptic feedback in a Mixed-Reality Environment. We then presented other applications that allows interaction with some kinds of non-rigid objects.

Finally, in chapter 8, we proposed a study on applications that do not necessarily need a realistic interaction paradigm. We conducted two experiments in the field of teleoperation. The first one consisted in driving a small robot. We performed a study on four interfaces that combines visual feedback with two-handed 3D interaction. Then, we proposed a second teleoperation experiment using an aerial robot.

9.2 Contributions

9.2.1 Two-Handed Haptic Feedback Rendering

The MHaptic framework embeds the research undergone to create a software haptic renderer for two-handed devices. First, we present a method to improve the data access rate. We propose a study, that could be reproduced for any haptic devices, whose goal is to evaluate the maximal refresh rate obtainable according to the link between the device and the computer. We then show how to take benefit of multithreaded approaches to optimize this rate.

We also propose methods to calibrate a two-handed device according to the applications that could take advantage from them. We show that it is precise enough to achieve realistic manipulation, but also quick and easy enough to rapidly change user.

We then suggest a method to improve user comfort. This method is applicable on any passive device that do not fully support user limbs. In this study, we also propose a novel method to evaluate the fatigue and use it to validate our work.

Considering the massive use of multicore CPUs in today's computers, we present the organization of a multithreaded haptic library. The structure of the library separates the main components into different threads. We briefly present the optimized synchronization that eases the communication between them. We put emphasis on a technique made to efficiently render force feedback on the two hands without sacrificing visual quality.

Finally, we present an effective software that allows to edit visual models of complex Virtual Environments. We present some useful techniques that avoid tedious tasks, and show that this software makes the addition of haptic properties to a virtual object very easy.

The results of these contributions have been accepted and published in international conference proceedings, particularly in [89] and [88].

9.2.2 Applications of Two-handed Haptics in Virtual Reality

In the second part of the thesis, we focused on two-handed haptic feedback and 3D interactions in various contexts. We created a first assembly training application in a Mixed-Reality Environment in which the user is able to assemble and interact with real and virtual objects. We proposed a feasibility study that allows to perform a test on the efficiency of different kinds of realistic interfaces. We present interesting results about precision, and propose useful recommendations about the design of such systems. We demonstrate that, in this context, the haptic guidance technique presents advantages. We then show that the MHaptic framework and the NVIDIA PhysX engine are also suitable for deformable objects manipulation. It proves that physics engine integration is a good choice for creating a haptic library.

We also proposed the integration of two teleoperation interfaces. We show that a two-handed haptic device is a promising technique to remote control any kind of robots. Even if it is not as efficient as a dedicated system, a two-handed device is generic, and thus is able to control many robots. Furthermore, we also provide an interesting conclusion: it is better to create new interaction paradigms rather than simulating a real interface. We prove with a validation that the simulation of a real interface (like a steering wheel) is less efficient than a gesture-based interface for example. We can thus provide two possible conclusions: either the device itself is not precise/efficient/actuated enough to simulate a steering wheel, or a real steering wheel is not the best tool to handle a vehicle. We believe it is perhaps a mix of both.

These contributions have been published in international journals [92] [93], and conference proceedings [90] [91] [57].

9.3 Perspectives

9.3.1 Results Improvement

The haptic rendering engine resulting from this thesis combines a realistic force feedback perception with a realistic animation of manipulated virtual objects, and this in generic Virtual Environments. However, some aspects of our method could be improved.

In terms of haptic rendering, there are some issues that could be addressed. One of the first that comes in mind, is the texture rendering. Even if the mass spring system is able to slightly render this effect based on the friction coefficient of the objects, it is not really the best approach. We believe that if a texture is applied on an object and taken into account by the force feedback engine, it would enhance the perception of the nature of it, even if the Haptic Workstation$^{\mathrm{TM}}$ is not the best tool to simulate this effect.

Another issue is related to the use of the PhysX library, and the fact that it is not open source. The library does not provide access to all information, although we know that this information has been computed. This force us to use some tricks and to approximate the information. We hope that PhysX will become open source or that an equivalent of the same quality will be issued in the near future. Hopefully, it is in fact very probable since Physics engine providers are currently competing to gain the position of leader.

Then, as mentioned in section 6.1, we remarked that users of the system do not perform equally with the same parametrization of the spring hand model. The study of this fact has not been addressed in this thesis.

Many users put also the finger on the fact that the arms are not displayed. We never add the forearm to the simulation because we do not exactly know its posture. However, it is probable that even if we display it in the bad position, the immersion will be greater.

Finally, concerning the applications, many more studies should be undertaken in order to validate the fact that the use of two hands presents advantages. We already present some, but we think that, due to the potential of the concept, the study of a single application could almost be the subject of a single thesis.

9.3.2 The Future of two-handed Haptics

As mentioned earlier in this conclusion, we believe that two-handed Haptics has a promising future. The intuitiveness and efficiency resulting from the use of such technologies for the manipulation of Virtual Environments allow us to think that two-handed devices will become the new standard of Haptics in few years. We are conscious that, in 2008, such devices have a very high cost. However, situation was the same for the Phantom® 10 years ago, and today, some devices that imitate it are commercially available for less than 300 dollars.

However, if we had to choose only one improvement of the Haptic Workstation$^{\mathrm{TM}}$, it would be the addition of a device stimulating the palm. This is one of the most common remarks that we had from the dozens of users of the device. It is true that the grasping state usually provokes a pressure on a part of the palm. We believe that a study on the importance of the palm pressure feeling could provide interesting results.

Finally, in the same state of mind, the adjunction of tactile devices on the fingertips would greatly increase the discrimination of shapes. It is really challenging to combine these two haptic perceptions, but it has already be done on a smaller scale with success.

Bibliography

[1] R. J. Adams and B. Hannaford. Control law design for haptic interfaces to virtual reality. *IEEE Transactions on Control Systems Technilogy*, 10(1):3–13, January 2002.

[2] D. L. Akin, M. L. Minsky, E. D. Thiel, and C. R. Kurtzman. *Space Applications of Automation, Robotics and Machine Intelligence Systems (ARAMIS)-Phase II*, volume 1. NASA Contractor Report 9734, prepared for NASA Marshall Space Flight Center, 1983.

[3] D. Avis and D. Bremner. How good are convex hull algorithms? In *SCG '95: Proceedings of the eleventh annual symposium on Computational geometry*, pages 20–28, New York, NY, USA, 1995. ACM.

[4] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality. *IEEE Computer Graphics Application*, 6:34–47, 2001.

[5] C. L. Bajaj and T. K. Dey. Convex decomposition of polyhedra and robustness. *SIAM J. Comput.*, 21(2):339–364, 1992.

[6] S. Bandi and D. Thalmann. An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies. *Computer Graphics Forum*, 14(3):259–279, 1995.

[7] D. Baraff and A. Witkin. Physically based modeling: Principles and practice. In *Online Siggraph '97 Course notes*. ACM Press, 1997.

[8] F. Barbagli, J. Salisbury, K., and R. Devengenzo. Enabling multi-finger, multi-hand virtualized grasping. *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, 1:809–815 vol.1, Sept. 2003.

[9] F. Barbagli and K. Salisbury. The effect of sensor/actuator asymmetries in haptic interfaces. *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2003. HAPTICS 2003. Proceedings. 11th Symposium on*, pages 140–147, March 2003.

[10] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[11] C. Basdogan, C.-H. Ho, and M. Srinivasan. Virtual environments for medical training: graphical and haptic simulation of laparoscopic common bile duct exploration. *Mechatronics, IEEE/ASME Transactions on*, 6(3):269–285, Sep 2001.

[12] M. Benali-Khoudja, M. Hafez, J. Alexandre, and A. Kheddar. Tactile interfaces: a state-of-the-art survey. In *proceedings of the 2004 International Symposium on Robotics*, 2004.

[13] M. Benali-Khoudjal, M. Hafez, J.-M. Alexandre, J. Benachour, and A. Kheddar. Thermal feedback model for virtual reality. *Micromechatronics and Human Science, 2003. MHS 2003. Proceedings of 2003 International Symposium on*, pages 153–158, Oct. 2003.

[14] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[15] M. Bergamasco, P. Degl'Innocenti, and D. Bucciarelli. A realistic approach for grasping and moving virtual objects. *Intelligent Robots and Systems '94. 'Advanced Robotic Systems and the Real World', IROS '94. Proceedings of the IEEE/RSJ/GI International Conference on*, 1:717–724 vol.1, Sep 1994.

[16] P. Berkelman, R. Hollis, and D. Baraff. Interaction with a real time dynamic environment simulation using a magnetic levitation haptic interface device. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 4:3261–3266 vol.4, 1999.

[17] G. Bianchi, B. Knoerlein, G. Szekely, and M. Harders. High precision augmented reality haptics. In *proceedings of the EuroHaptics Conference*, 2006.

[18] A. Boeing and T. Bräunl. Evaluation of real-time physics simulation systems. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 281–288, New York, NY, USA, 2007. ACM.

[19] S. Booth, F. Angelis, and T. Schmidt-Tjarksen. The influence of changing haptic refresh-rate on subjective user experiences - lessons for effective touch-based applications. In *proceedings of the 2003 Eurohaptics conference*, pages 374–383, 2003.

[20] C. W. Borst and A. P. Indugula. Realistic virtual grasping. In *proceedings of the 2005 IEEE Conference 2005 on Virtual Reality (VR'05)*, pages 91–98, 320, Washington, DC, USA, 2005. IEEE Computer Society.

[21] C. W. Borst and A. P. Indugula. A spring model for whole-hand virtual grasping. *Presence : Teleoperators and Virtual Environments*, 15(1):47–61, February 2006.

[22] R. Boulic, S. Rezzonico, and D. Thalmann. Multi-finger manipulation of virtual objects. In *ACM Symposium on Virtual Reality and Technology*, 1996. Comput. Graphics Lab., Fed. Inst. of Technol., Lausanne, Switzerland.

[23] R. Brown, S. Schneider, and M. Mulligan. Analysis of algorithms for velocity estimation from discrete position versus time data. *IEEE Transactions on Industrial Electronics*, 39(1):11–19, Feb 1992.

[24] S. Bruce, C. Rice, and R. Hepp. Design and test of military cockpits. In *Proceedings of IEEE Aerospace Conference*, volume 3, pages 5–14, 1998.

[25] N. Bu, O. Fukuda, and T. Tsuji. EMG-Based motion discrimination using a novel recurrent neural network. *Journal of Intelligent Information Systems (JIIS)*, 21(2):113–126, 2003.

[26] E. Burns, S. Razzaque, A. T. Panter, M. C. Whitton, M. R. McCallus, and J. Frederick P. Brooks. The hand is more easily fooled than the eye: Users are more sensitive to visual interpenetration than to visual proprioceptive discrepancy. *Presence : Teleoperators and Virtual Environments*, 15(1):1–15, February 2006.

[27] P. Buttolo, A. Marsan, and P. Stewart. A haptic hybrid controller for virtual prototyping of vehicle mechanisms. In *HAPTICS '02: Proceedings of the 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, page 249, Washington, DC, USA, 2002. IEEE Computer Society.

[28] W. A. S. Buxton. Chunking and phrasing and the design of human-computer dialogues. pages 494–499, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[29] Y. Cai, S. Wang, and M. Sato. A human-scale direct motion instruction system device for education systems. *IEICE TRANSACTIONS on Information and Systems*, 80(2):212–217, 1997.

[30] A. E. Çakir, G. Çakir, T. Müller, and P. Unema. The trackpad: a study on user comfort and performance. In *Conference companion on Human factors in computing systems*, pages 246–247. ACM Press, 1995.

[31] D. Checcacci, E. Sotgiu, A. Frisoli, C. Avizzano, and M. Bergamasco. Gravity compensation algorithms for parallel haptic interface. In *Proceedings of IEEE International Workshop on Robot and Human Interactive Communication, Berlin, Germany*, pages 140–145, 2002.

[32] H. Chen and H. Sun. Real-time haptic sculpting in virtual volume space. In *Proceedings of the 2002 ACM symposium on Virtual reality software and technology*, pages 81–88, New York, NY, USA, 2002. ACM.

[33] F. Conti, F. Barbagli, D. Morris, and C. Sewell. Chai: An open-source library for the rapid development of haptic scenes. In *Demo paper presented at IEEE World Haptics*, March 2005.

[34] F. Crison, A. Lécuyer, D. Mellet-D'Huart, J. M. Burkhardt, G. Michel, and J. L. Dautin. Virtual technical trainer: Learning how to use milling machines with multi-sensory feedback in virtual reality. In *proceedings of the IEEE International Conference on Virtual Reality (VR'05)*, 2005.

[35] C. Darwin. *The Descent of Man, and Selection in Relation to Sex*. Princeton University Press, 1871.

[36] R. V. der Linde, P.Lammertse, E. Frederiksen, and B. Ruiter. The hapticmaster, a new high-performance haptic interface. In *proceedings of the 2002 Eurohaptics Conference*, 2002.

[37] J. E. Deutsch, J. Latonio, G. C. Burdea, and R. Boian. Post-stroke rehabilitation with the rutgers ankle system: A case study. *Presence: Teleoper. Virtual Environ.*, 10(4):416–430, 2001.

[38] J. Dionisio. Virtual hell: a trip through the flames. *Computer Graphics and Applications, IEEE*, 17(3):11–14, May/Jun 1997.

[39] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra: a unified approach. In *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, pages 400–413, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[40] L. Dominjon, A. Lécuyer, J. Burkhardt, G. Andrade-Barroso, and S. Richir. The "bubble" technique: Interacting with large virtual environments using haptic devices with limited workspace. In *proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WORLDHAPTICS'05)*.

[41] L. Dominjon, S. Richir, A. Lécuyer, and J.-M. Burkhardt. Haptic hybrid rotations: Overcoming hardware angular limitations of force-feedback devices. In *VR '06: Proceedings of the IEEE conference on Virtual Reality*, pages 164–174, Washington, DC, USA, 2006. IEEE Computer Society.

[42] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufmann, 2005.

[43] J. Fernandez-Lozano, J. M. G. de Gabriel, V. F. Munoz, I. Garcia-Morales, D. Melgar, C. Vara, and A. Garcia-Cerezo. Human-machine interface evaluation in a computer assisted surgical system. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 231–236. IEEE Computer Society Press, 2004.

[44] M. Fischer, P. van der Smagt, and G. Hirzinger. Learning techniques in a dataglove based telemanipulation system for the DLR hand. In *transactions of the IEEE International Conference on Robotics and Automation*, pages 1603–1608, 1998.

[45] M. Foskey, M. A. Otaduy, and M. C. Lin. Artnova: touch-enabled 3d model design. In *Proceedings of the 2002 IEEE Virtual Reality Conference*, pages 119–126, 2002.

[46] J. H. Freidman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[47] A. Frisoli, F. Rocchi, S. Marcheschi, A. Dettori, F. Salsedo, and M. Bergamasco. A new force-feedback arm exoskeleton for haptic interaction in virtual environments. *Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint*, pages 195–201, March 2005.

[48] A. Frisoli, M. Solazzi, F. Salsedo, and M. Bergamasco. A fingertip haptic display for improving curvature discrimination. *Presence: Teleoper. Virtual Environ.*, 17(6):550–561, 2008.

[49] P. Fuchs and G. Moreau. *Le Traité de la Réalité Virtuelle, deuxième édition*, volume 1. Les Presses de l'Ecole des Mines de Paris, 2004.

[50] D. G.Caldwel1, O.Kocak, and U.Andersen. Multi-armed dexterous manipulator operation using glove/exoskeleton control and sensory feedback. In *IROS '95: Proceedings of the International Conference on Intelligent Robots and Systems-Volume 2*, page 2567, Washington, DC, USA, 1995. IEEE Computer Society.

[51] J. Goble, K. Hinckley, R. Pausch, J. Snell, and N. Kassell. Two-handed spatial interface tools for neurosurgical planning. *Computer*, 28(7):20–26, Jul 1995.

[52] R. C. Goertz and W. M. Thompson. Electronically controlled manipulator. *Nucleonics*, 12(11):46, 1954.

[53] G. H. Gollub and C. F. V. Loan. *Matrix Computations*, chapter 8.4, pages 426–429. The Johns Hopkins University Press, 1996.

[54] W. B. Griffin, R. P. Findley, M. L. Turner, and M. R. Cutkosky. Calibration and mapping of a human hand for dexterous telemanipulation. In *proceedings of the Haptic Interfaces for Virtual Environments and Teleoperator Systems Symposium*, 2000.

[55] Y. Guiard. Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model. *Journal of Motor Behavior*, 19:486–517, 1987.

[56] M. Gutiérrez, P. Lemoine, D. Thalmann, and F. Vexo. Telerehabilitation: controlling haptic virtual environments through handheld interfaces. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 195–200, New York, NY, USA, 2004. ACM.

[57] M. Gutierrez, R. Ott, D. Thalmann, and F. Vexo. Mediators: Virtual haptic interfaces for tele-operated robots. In *Proceedings of the 13th IEEE International Workshop on Robot and Human Interactive Communication (RO-MAN'04)*, pages 515–520, 2004.

[58] C. Hand. A survey of 3d interaction techniques. *COMPUTER GRAPHICS forum*, 16(5):269–281, 1997.

[59] H. J. Haverkort. *Results on geometric networks and data structures*. PhD thesis, Utrecht University, 2004.

[60] V. Hayward and O. R. Astley. Performance measures for haptic interfaces. In *Robotics Research: The 7th International Symposium*, pages 195–207. Springer Verlag, 1996.

[61] V. Hayward, O. R. Astley, M. Cruz-Hernandez, D. Grant, and G. Robles-De-La-Torre. Haptic interfaces and devices. *Sensor Review*, 24(1):16–29, 2004.

[62] Y. Hirata and M. Sato. 3-dimensional interface device for virtual work space. *Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on*, 2:889–896, Jul 1992.

[63] B. J. Holbert. *Enhanced targeting in a haptic user interface for the physically disabled using a force feedback mouse*. PhD thesis, Arlington, TX, USA, 2007. Adviser : Manfred Huber.

[64] J. M. Hollerbach, E. Cohen, W. Thompson, R. Freier, D. Johnson, A. Nahvi, D. Nelson, and T. V. T. Ii. Haptic interfacing for virtual prototyping of mechanical cad designs. In *In ASME Design for Manufacturing Symposium*, pages 14–17. Kluwer Academic Publishers, 1997.

[65] H. Iwata. Artificial reality with force-feedback: development of desktop virtual space with compact master manipulator. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 165–170, New York, NY, USA, 1990. ACM.

[66] H. Iwata, H. Yano, T. Uemura, and T. Moriya. Food simulator: A haptic interface for biting. *Virtual Reality Conference, IEEE*, 0:51, 2004.

[67] I. T. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[68] L. A. Jones and M. Berris. The psychophysics of temperature perception and thermal-interface design. In *HAPTICS '02: Proceedings of the 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, page 137, Washington, DC, USA, 2002. IEEE Computer Society.

[69] F. Kahlesz, G. Zachmann, and R. Klein. Visual-fidelity dataglove calibration. In *proceedings of Computer Graphics International*, pages 403–410, 2004.

[70] S. Kim, S. Hasegawa, Y. Koike, and M. Sato. Tension based 7-dof force feedback device: Spidar-g. *Virtual Reality, 2002. Proceedings. IEEE*, pages 283–284, 2002.

[71] E. Kreighbaum and K. M. Barthels. *Biomechanics: A Qualitative Approach for Studying Human Movement*, chapter Appendix III. Burgess Pub. Co, 1983.

[72] W. Kuo-Cheng, T. Fernando, and H. Tawfik. Freesculptor: a computer-aided freeform design environment. In *Proceedings of the 2003 International Conference on Geometric Modeling and Graphics*, pages 188–194, Juillet 2003.

[73] A. Lécuyer, J. Burkhardt, J. L. Biller, and M. Congedo. A^4: A technique to improve perception of contacts with under-actuated haptic devices in virtual reality. In *proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WORLDHAPTICS'05)*, March 2005.

[74] A. Lécuyer, A. Kheddar, S. Coquillart, L. Graux, and P. Coiffet. A haptic prototype for the simulations of aeronautics mounting/unmounting operations. In *proceedings of the IEEE International Workshop on Robot-Human Interactive Communication (RO-MAN'01)*, 2001.

[75] C. Lee, M. S. Hong, I. Lee, O. K. Choi, K.-L. Han, Y. Y. Kim, S. Choi, and J. S. Lee. Mobile haptic interface for large immersive virtual environments: Pomhi v0.5. *Journal of Korea Robotics Society*, 3(2), 2008.

[76] C. Lee and Y. Xu. Online, interactive learning of gestures for human/robot interfaces. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 2982–2987, 1996.

[77] P. Lemoine, M. Gutierrez, F. Vexo, and D. Thalmann. Mediators: Virtual interfaces with haptic feedback. In *Proceedings of EuroHaptics 2004, 5th-7th June, Munich, Germany*, pages 68–73, 2004.

[78] M. Lin and J. Canny. A fast algorithm for incremental distance calculation. *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1008–1014 vol.2, 9-11 Apr 1991.

[79] D. Mellet-d'Huart, G. Michela, J. M. Burkhardt, A. Lécuyer, J. L. Dautin, and F. Crison. An application to training in the field of metal machining as a result of research-industry collaboration. In *proceedings of the Virtual Reality Conference (VRIC)*, 2004.

[80] T. Miller and R. Zeleznik. An insidious haptic invasion: adding force feedback to the x desktop. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 59–64, New York, NY, USA, 1998. ACM.

[81] B. Mirtich. V-clip: fast and robust polyhedral collision detection. *ACM Trans. Graph.*, 17(3):177–208, 1998.

[82] T. Möller. A fast triangle-triangle intersection test. *J. Graph. Tools*, 2(2):25–30, 1997.

[83] D. G. R. C. M.R. Tremblay, C. Ullrich and J. Tian. Whole-hand interaction with 3d environments. Technical report, Virtual Technologies Inc., 1997.

[84] J. Murayama, L. Bougrila, Y. K. Akahane, S. Hasegawa, B. Hirsbrunner, and M. Sato. Spidar g&g: A two-handed haptic interface for bimanual vr interaction. In *Proceedings of EuroHaptics 2004*, pages 138–146, 2004.

[85] B. Naylor, J. Amanatides, and W. Thibault. Merging bsp trees yields polyhedral set operations. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 115–124, New York, NY, USA, 1990. ACM.

[86] T. Nojima, D. Sekiguchi, M. Inami, and S. Tachi. The smarttool: a system for augmented reality of haptics. In *proceedings of the IEEE Virtual Reality Conference*, pages 67–72, 2002.

[87] M. A. Otaduy and M. C. Lin. *High Fidelity Haptic Rendering*, chapter 6 DOF Haptic Rendering Methodologies, pages 23–34. Morgan and Claypool Publishers, 2006.

[88] R. Ott, V. de Perrot, D. Thalmann, and F. Vexo. Mhaptic: a haptic manipulation library for generic virtual environments. In *Proceedings of the Cyberworlds 2007 International Conference*, pages 338–345, 2007.

[89] R. Ott, M. Gutierrez, D. Thalmann, and F. Vexo. Improving user comfort in haptic virtual environments trough gravity compensation. In *proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WORLDHAPTICS'05)*, pages 401–409, 2005.

[90] R. Ott, M. Gutierrez, D. Thalmann, and F. Vexo. Vr haptic interfaces for teleoperation : an evaluation study. In *Proceedings of the IEEE Intelligent Vehicles Symposium, (IV'05)*, 2005.

[91] R. Ott, M. Gutierrez, D. Thalmann, and F. Vexo. Advanced virtual reality technologies for surveillance and security applications. In *Proceedings of the ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications (VRCIA'06)*, 2006.

[92] R. Ott, D. Thalmann, and F. Vexo. Organic shape modelling. *Computer-Aided Design and Applications*, 3(1–4):79–88, 2006.

[93] R. Ott, D. Thalmann, and F. Vexo. Haptic feedback in mixed-reality environment. *The Visual Computer*, 23(9-11):843–849, 2007.

[94] A. Peternier, D. Thalmann, and F. Vexo. Mental vision: a computer graphics teaching platform. In *proceedings of the 2006 Edutainment Conference*, pages 223–232, 2006.

[95] A. Peternier, F. Vexo, and D. Thalmann. The mental vision framework: a platform for teaching, practicing and researching with computer graphics and virtual reality. *LNCS Transactions on Edutainment*, 2008.

[96] ReachIn API, Touch Enabled Solutions. http://www.reachin.se/products/reachinapi/.

[97] B. Runck. What is Biofeedback?. DHHS Publication No (ADM) 83-1273, Arizona Behavioral Health Associates, P.C. http://www.psychotherapy.com/bio.html.

[98] D. C. Ruspini, K. Kolarov, and O. Khatib. The haptic display of complex graphical environments. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 345–352, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[99] P. Salamin, D. Thalmann, and F. Vexo. Comfortable manipulation of a virtual gearshift prototype with haptic feedback. In *Proceedings of the 2006 ACM symposium on Virtual reality software and technology*, 2006.

[100] H. Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, 1984.

[101] Schuhfried GmbH. Physiorecorder.
http://www.schuhfried.co.at.

[102] C. S. Sherrington. On the proprioceptive system, especially in its reflex aspect. *Brain*, 29:497–482, 1907.

[103] M. Slater and A. Steed. A virtual presence counter. *Presence : Teleoperators and Virtual Environments*, 9(5):413–434, October 2000.

[104] S. S. Snibbe, K. E. MacLean, R. Shaw, J. Roderick, W. L. Verplank, and M. Scheeff. Haptic techniques for media control. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 199–208, New York, NY, USA, 2001. ACM.

[105] A. Soares, A. Andrade, E. Lamounier, and R. Carrijo. The development of a virtual myoelectric prosthesis controlled by an emg pattern recognition system based on neural networks. *J. Intell. Inf. Syst.*, 21(2):127–141, 2003.

[106] H. Sowizral. Scene graphs in the new millennium. *IEEE Computer Graphics and Applications*, 20(1):56–57, 2000.

[107] D. J. Sturman and D. Zeltzer. A survey of glove-based input. *IEEE Comput. Graph. Appl.*, 14(1):30–39, 1994.

[108] Z. Sun, G. Bao, J. Li, and Z. Wang. Research of dataglove calibration method based on genetic algorithms. In *proceedings of the 6th World Congress on Intelligent Control and Automation*, pages 9429–9433, 2006.

[109] I. E. Sutherland. The ultimate display. In *Proceedings of IFIPS Congress*, volume 2, pages 506–508, 1965.

[110] H. Tamura, Yamamoto, and A. Katayama. Mixed reality: future dreams seen at the border between real and virtual worlds. *IEEE Computer Graphics and Applications*, 21(6):64–70, November/December 2001.

[111] N. Tarrin, S. Coquillart, S. Hasegawa, L. Bouguila, and M. Sato. The stringed haptic workbench. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Sketches & Applications*, pages 1–1, New York, NY, USA, 2003. ACM.

[112] The Engineering Toolbox, Friction and Coefficient of Friction for some common material combinations. http://www.engineeringtoolbox.com/friction-coefficients-d_778.html.

[113] M. Tuceryan and N. Navab. Single point active alignment method (spaam) for optical see-through hmd calibration for ar. In *proceedings of the IEEE and ACM International Symposium on Augmented Reality (ISAR'2000)*, pages 149–158, 2000.

[114] M. Ueberle and M. Buss. Control of kinesthetic haptic interfaces. In *Proceedings of IEEERSJ International Conference on Intelligent Robots Systems, Workshop Touch Haptics*, pages 147–151, 2004.

[115] L. Vacchetti, V. Lepetit, M. Ponder, G. Papagiannakis, D. Thalmann, N. Magnenat-Thalmann, and P. Fua. *A Stable Real-Time AR Framework for Training and Planning in Industrial Environments*, pages 129–146. Springer, 2004.

[116] S. Walairacht, K. Yamada, S. Hasegawa, Y. Koike, and M. Sato. 4 + 4 fingers manipulating virtual objects in mixed-reality environment. *Presence: Teleoperators and Virtual Environments*, 11(2):134–143, 2002.

[117] D. Xu. A neural network approach for hand gesture recognition in virtual reality driving training system of spg. In *Proceedings of the 18th IEEE International Conference on Pattern Recognition (ICPR'06)*, pages 519–522, 2006.

[118] G.-H. Yang, K.-U. Kyung, M. A. Srinivasan, and D.-S. Kwon. Development of quantitative tactile display device to provide both pin- array-type tactile feedback and thermal feedback. In *WHC '07: Proceedings of the Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 578–579, Washington, DC, USA, 2007. IEEE Computer Society.

[119] J. Zauner, M. Haller, A. Brandl, and W. Hartman. Authoring of a mixed reality assembly instructor for hierarchical structures. In *proceedings of the Second IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 237–246, October 2003.

[120] C. B. Zilles and J. K. Salisbury. A constraint-based god-object method for haptic display. In *proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 146–151, 1995.

Appendices

Appendix I: The Haptic Scene file format

This appendix presents the *Document Type Definition* of a XML Haptic File.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--
 A Haptic scene decription document type (for Ageia).
 The body is not saved because it is not necessary:
 - If the actor is Dynamic, a body is associated during the
     creation of the object and computed
 from its shapes: inertia tensor, center of mass etc.
 - If the actor is Static, no body is created.
 - The Scene is constituted by Object
 - The Objects are either Dynamic or Static and this
     information is an attribute:
   * An Object must first have a name
   * Then it has a global position (position of the
       coordinate system of the object)
   * Then it has a global rotation (rotation of the
       coordinate system of the object)
   * Then it has at least one Shape with an attribute
       describing its type
     + A Shape has a local position (position relative to
         the coordinate system of the object)
     + A Shape has a local rotation (rotation relative to
         the coordinate system of the object)
     + A Shape has either a Density or a Mass
       (2 notes: - When mass <=0.0 then density and volume
          determine the mass
                 - When object is Static, this information
                    is not used at all)
     + If attribute is plane, the Shape has then 4
         parameters (a,b,c,d for ax+by+cz=d)
```

```
      + If attribute is sphere, the Shape has then 1
         parameter (the radius)
      + If attribute is box, the Shape has then 3 parameters
         (half the sides)
      + If attribute is Capsule, the Shape has then 2
         parameters (radius and length)
      + (not supported yet) If attribute is a Trimesh (one
         fake param)
 – Mass is a number [kg]
 – Density is a number [unit = kg.m–3 , Default value is
    1.0]
 – Position is 3 numbers [m]
 – Rotation is 3x3 numbers (heavy but straighforward)
 – A number is either a float, a double or an integer
—>

<!ENTITY % number "integer|float|double">
<!ELEMENT integer (#PCDATA)>
<!ELEMENT float (#PCDATA)>
<!ELEMENT double (#PCDATA)>
<!ELEMENT MHAPTIC:scene (MHAPTIC:object)*>
<!ATTLIST MHAPTIC:scene xmlns:MHAPTIC CDATA #REQUIRED>
<!ELEMENT MHAPTIC:object (MHAPTIC:name, MHAPTIC:position,
   MHAPTIC:rotation, (MHAPTIC:shape)+)>
<!ATTLIST MHAPTIC:object type (Static | Dynamic) #REQUIRED>
<!ELEMENT MHAPTIC:shape (MHAPTIC:position, MHAPTIC:rotation,
    (MHAPTIC:density | MHAPTIC:mass), (MHAPTIC:planeparams |
    MHAPTIC:sphereparams | MHAPTIC:boxparams |
   MHAPTIC:capsuleparams | MHAPTIC:trimeshparams))>
<!ATTLIST MHAPTIC:shape type (Plane | Sphere | Box | Capsule
    | Trimesh) #REQUIRED>
<!ELEMENT MHAPTIC:position ((%number;),(%number;),(%number;)
   )>
<!ELEMENT MHAPTIC:rotation ((%number;),(%number;),(%number;)
   ,(%number;),(%number;),(%number;),(%number;),(%number;)
   ,(%number;))>
<!ELEMENT MHAPTIC:density (%number;)>
<!ELEMENT MHAPTIC:mass (%number;)>
<!ELEMENT MHAPTIC:planeparams ((%number;), (%number;), (%
   number;), (%number;))>
<!ELEMENT MHAPTIC:sphereparams (%number;)>
<!ELEMENT MHAPTIC:boxparams ((%number;), (%number;), (%
   number;))>
<!ELEMENT MHAPTIC:capsuleparams ((%number;), (%number;))>
<!ELEMENT MHAPTIC:trimeshparams (%number;)>
```

```
<!ELEMENT MHAPTIC:name (#PCDATA)>
```

This an example of a simple scene:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE MHAPTIC:scene SYSTEM "HapticScene.dtd">
<MHAPTIC:scene xmlns:MHAPTIC="http://vrlab.epfl.ch/~reno">
  <MHAPTIC:object type="Static">
    <MHAPTIC:name>groundLiving</MHAPTIC:name>
    <MHAPTIC:position>
      <float>1.766357</float>
      <float>-0.000000</float>
      <float>-4.121672</float>
    </MHAPTIC:position>
    <MHAPTIC:rotation>
      <float>1.000000</float>
      <float>0.000000</float>
      <float>0.000000</float>
      <float>0.000000</float>
      <float>1.000000</float>
      <float>0.000000</float>
      <float>0.000000</float>
      <float>0.000000</float>
      <float>1.000000</float>
    </MHAPTIC:rotation>
    <MHAPTIC:shape type="Plane">
      <MHAPTIC:position>
        <float>0.000000</float>
        <float>0.000000</float>
        <float>0.000000</float>
      </MHAPTIC:position>
      <MHAPTIC:rotation>
        <float>1.000000</float>
        <float>0.000000</float>
        <float>0.000000</float>
        <float>0.000000</float>
        <float>1.000000</float>
        <float>0.000000</float>
        <float>0.000000</float>
        <float>0.000000</float>
        <float>1.000000</float>
      </MHAPTIC:rotation>
      <MHAPTIC:materialparams>
        <float>0.240000</float>
        <float>0.350000</float>
        <float>0.200000</float>
      </MHAPTIC:materialparams>
      <MHAPTIC:planeparams>
        <float>0.000000</float>
        <float>1.000000</float>
        <float>0.000000</float>
        <float>−0.000001</float>
      </MHAPTIC:planeparams>
    </MHAPTIC:shape>
  </MHAPTIC:object>
</MHAPTIC:scene>
```

Appendix II: Programming with MHaptic

This appendix presents the code of a simple program that uses MHaptic. Some parts have been removed because they are not useful to understand the library. We can see that a program should start by initializing the visual Virtual Environment using MVisio. The visual scene is stored in a `.mve` file. Then, we need to initialize MHaptic and to add the haptic information to the objects using `MHAPTIC::loadScene(...)`. Then, the only thing to do is to render as fast as possible the visual scene. The updates of force, objects position, hand posture are embedded into MHaptic.

The listing:

```
#include <mhaptic/mhaptic.h>

int main(int argc, char *argv[]) {

   //Visual Display initialization
   MVSETTINGS settings;
   settings.setColorDepth(32);
   settings.setWindowX(640);
   settings.setWindowY(480);
   settings.setFullScreen(false);
   settings.setTitle("MHAPTIC␣Simple␣Program");
   if (MVISIO::init(&settings) == false) {
      printf("Display␣Aborted\n");
      fflush(stdout);
      return 0;
   }

   // Visual Virtual Environment Initialization
   MVCAMERA * camera = new MVCAMERA();
   camera->setPosition(0.5, 0.0, 0.5);
   camera->setTarget(0.0 , 0.0 ,0.0);

   MVLIGHT * light = new MVLIGHT();
   light->setPosition(5.0, 10.0, 0.0);
```

```
MVNODE * scene = MVISIO::load("MyTestScene.mve");

//Haptic initialization
MHAPTIC::init();
//Connect with every devices
MHAPTICWORKSTATION::connectAll();

// Start Haptic Thread
MENGINE::startHapticThread();

// Add Haptic Information to the already loaded Visual
   Scene
MHAPTIC::loadScene("MyTestScene.xml");

bool done = false;

while(!done){

   // Check Keyboard Events:
   char key = 0;
   SDL_Event event;
   while (SDL_PollEvent(&event)) {
      if (event.type == SDL_QUIT)
         done = 1;
      if (event.type == SDL_KEYDOWN) {
         if (event.key.keysym.sym == SDLK_ESCAPE)
            done = 1;
      }
   }
   SDL_PumpEvents();

   // check Mouse Events
   MVID buttonPressed = 0;
   static int oldMouseX, oldMouseY, deltaMouseX,
      deltaMouseY;
   int MouseX = 0;
   int MouseY = 0;
   int flags = SDL_GetMouseState(&MouseX, &MouseY);
   bool MBL = false, MBR = false;
   if (flags & SDL_BUTTON(1)) MBL = true;
   if (flags & SDL_BUTTON(3)) MBR = true;
```

```
        // MVisio 2D GUI events
        MVELEMENT::manageGUIEvents(key, MouseX, MouseY,
            MBL, MBR, &buttonPressed);

        // Forward Events to the MHPATIC GUI
        MHAPTIC::manageInterfaceEvent(buttonPressed);

        // Synchronize visual nodes pose using haptic nodes
        MHNODE::synchronizeAllPoses();

        // Updates the Visual Hand
        MHAPTIC::updateHandsPosition();

        // Make Visual Rendering of the Virtual Environment
        MVISIO::clear(true, true, true);
        MVISIO::begin3D(camera);
            light->pass();
            scene->pass();
            MHAPTIC::passHands(true, true, false);
        MVISIO::end3D();


        // Make 2D GUI Rendering
        MVISIO::begin2D();
            MHAPTIC::passInterface();
        MVISIO::end2D();

        MVISIO::swap();
    }

    // Free Memory
    MHAPTIC::deinit();
    MVISIO::deinit();

    // exit
    return 0;
}
```

Appendix III: HSC User Manual

User Manual of the Haptic Scene Creator

Welcome, and congratulations. You have just bought a ticket to the world of the Haptic Scene Creator. With this tool, you will be able to visualize, augment and simulate a virtual environment in no time.

This tutorial introduces the Haptic Scene Creator step by step. It is recommended to follow the chapters in this order as some important concepts are explained in the first chapters. In this tutorial we will show you how to navigate in the virtual environment, add and modify the physical properties of the objects, save and load a haptic scene and how to run the simulation dynamically. Some performance hints are finally given at the end of this tutorial.

Getting started with the Haptic Scene Creator

This first tutorial is provided to make sure that all components are installed before launching the Haptic Scene Creator.

First install the Ageia SDK software (current version is 2.6.2). The *Core.exe* file must be installed before the *SystemSoftware.exe* package or it will not work as expected at runtime.

If you want to compile the source code of the Haptic Scene Creator, (or even the MHaptic library), be sure that the following directories are included to your programming environment:

Header files:

- ...\AGEIA PhysX SDK\v2.6.2\SDKs\Physics\include
- ...\AGEIA PhysX SDK\v2.6.2\SDKs\Foundation\include
- ...\AGEIA PhysX SDK\v2.6.2\SDKs\Cooking\include
- ...\AGEIA PhysX SDK\v2.6.2\SDKs\PhysXLoader\include

Libraries:

- ...\AGEIA PhysX SDK\v2.6.2\SDKs\lib\win32
- ...\AGEIA PhysX SDK\v2.6.2\Bin\win32

In the path of the system, verify that the following environment variable is defined as well:

- ...\AGEIA PhysX SDK\v2.6.2\Bin\win32

Once these libraries installed, you are ready to launch the Haptic Scene Creator.

Overview of the interface

This section introduces the Graphical User Interface (GUI) of the Haptic Scene Creator.

The GUI is split in two parts: the four different windows (or viewports) and the toolbar, which is on the right side. If you are familiar with Autodesk® 3DS Max for example, you should easily get used to this layout.

The top-left, top-right and bottom-left viewports represent respectively the left, top and front point of views. They use orthographic projection and wireframe rendering mode. The bottom right window renders the scene in perspective mode with textured and lit objects.

The HSC has four different viewports, but only one can be selected at a time. To activate one of the four viewports, click inside with the left mouse button. A yellow square will surround the new active window. Try to select the four windows one at a time.

When operations are performed, some messages are displayed to keep you aware of what happened. Regular messages appear in the small message line on the bottom left of the screen. Warning messages are displayed in a message box. The number of frames per second (FPS) is also displayed in the toolbar, on the bottom right of the screen. It is a good performance indicator, especially when you run the physic simulation.

The toolbar on the right side of the screen contains six subparts (not all are visible at this time):

- Load and Save: which allows the loading of virtual environments, and the loading and saving of the haptic information.
- Camera speed: modify the speed of the camera.
- Rendering: display the different elements in the viewports.
- Add geometries: allows to fit collision geometries to the 3D models.
- Materials: modify the material properties of an object.

The details of each subpart will be explained in the next tutorials.

Loading and saving data files

This subsection explains how to load and save data files in the Haptic Scene Creator. The commands used for this tutorial are in the toolbar, at the very top of it.

The first thing you need to load is an MVisio file (`.mve`) of the scene using the [Load MVE] button in the toolbar .Be sure that you use the last version of the plug-in when exporting your environment from Autodesk® 3DS Max. The loading process is usually fast but can eventually take some time depending on the complexity of the environment. This data contains the graphical information of the 3D models in the scene only. The Haptic Scene Creator does not save MVisio files as it does not need to modify the graphical data.

The second step of this tutorial is to load an `.xml` file containing the haptic data using the [Load XML] button in the toolbar. You can also use the shortcut CTRL+O on the keyboard. If you do not have one, do not worry as we will quickly show how to create one in the next tutorials. This file contains the haptic data only. When loading a haptic file, be sure that the `.mve` is already loaded or the objects will not be correctly linked.

To save the haptic data, use the [Save XML] button in the toolbar. You can also use the shortcut CTRL+S on the keyboard. The scene can be saved anywhere on the disk (no need to save it in the folder containing the `.mve` file) as the haptic data is completely kept separated from the visual data.

To exit the Haptic Scene Creator, use the [Exit] button in the toolbar.

In the next tutorial, we will show how to navigate in the environment using the mouse and the keyboard.

Navigation in the virtual environment

This subsection introduces the navigation concepts in the Haptic Scene Creator. The first step is to load an MVisio environment using the [Load MVE] button in the toolbar. Now that your 3D environment is loaded, lets take a look at the navigation. The principle is different if you navigate in a window using orthographic projection or perspective projection. We explain here both modes of navigation.

In an orthographic projection view, it is possible to translate the camera to the left, right, top and bottom directions by dragging the right mouse button. First select the top right viewport (top camera view) with the left mouse button. Click with the right button in the window and move the camera by holding the button pressed. Release the button to stop moving. Notice how the camera follows exactly the cursor of the mouse. You can also zoom in and zoom out using the mouse wheel. Zooming in allow you to see the details of an object. Zooming out gives you a global view of the environment. It is possible to zoom and translate the camera at any time, as long as the corresponding viewport is selected. The navigation works exactly the same way in the two other views using orthographic projections.

Let's take a look at the navigation in the perspective view, where the concept is slightly different. Please first select the bottom right window with the left button. To translate the camera left and right, use the left/right arrows on the keyboard or the *s*/*f* keys. To move up or down, use the *t* and *g* keys. To move forward and backward, do not use the wheel mouse, because this modifies the Field Of View (FOV). Use instead the up/down arrows or the *e*/*d* keys. Now you can move in the three translational directions (left/right, up/down, forward/backward). The rotation of the camera is performed by holding the right mouse button pressed and moving the mouse. To rotate the camera on the left, move the mouse to the left, same for other directions. Now try to move the camera with the keyboard and rotate it simultaneously using the mouse. If you are familiar with 3D applications or games using first-person cameras, you are already mastering it. If not, a little bit of practice will certainly help.

If you do not like the speed of the camera, you can modify it using the camera scrollbar in the toolbar.

Selecting an object is quite simple. Just clicking on it with the middle mouse button will select it. Be sure that the viewport is selected or the command will not be executed. The texture of a selected object is highlighted and can be recognized easily. Notice how the three orthographic views are automatically aligned on the object. If you deselect an object (click anywhere else), the camera will return to the previous position.

Vertex selection allows you to select only the subpart of an object when you want to fit a collision geometry (this is explained in the next tutorial). This is what makes the Haptic Scene Creator fast and easy to use. Vertex selection can be done in any viewport. An object must be selected to be able to select some of its vertices. First, select an object with the left mouse button. Then, draw a 2-dimensional rectangle with the left button around the vertices you want to select. Release the button when you the rectangle. The selected vertices will appear in red on the screen. You can repeat the operation as many times as needed.

Now that you know how to navigate in the environment, select an object and select some vertices, we are ready to add some haptic properties to the objects in the next tutorial.

Adding haptic properties to an object

In this tutorial we show how to add haptic properties to an object. An object will interact in the physical simulation only if an haptic structure is attached to it. If a 3D object in your scene does not have some haptic information, it will not take part to the simulation.

The haptic structure contains two kind of information: the mass of the rigid body and its collision geometries. Because the Haptic Scene Creator computes almost all information automatically concerning the mass (inertial tensor, center of mass . . .), all you need to do is to attach collision primitives and specify either a mass or a density to the body.

To add a collision geometry to an object, first select it with the middle mouse button. In the toolbar, the [Add geometry] window will appear. It contains all the buttons to add various collision geometries that fits automatically to the selected object.

An object is static by default. If you want to make it dynamic, click in the toolbar on the [make dynamic] button. You can now enter either the mass $[kg]$ or the density $[g/cm^3]$. Confirm with the [set mass] or [set density] buttons.

The different kinds of collision geometries are now presented in detail. Then, in the next tutorial we show how to modify the collision geometries with the arcball.

Boxes, spheres and capsules

The basic geometrical primitives are the following: boxes, spheres and capsules (also called line-swept sphere or capped cylinders). Most simple objects can be approximated by just using these shapes.

To add a box, a sphere or a capsule, first select an object and eventually select some vertices if you want to fit the geometry to a subpart of the object. Then, in the toolbar, use the buttons located in the [Add Geometry] window to fit various kinds of geometries. We present here the five first kinds of collision geometry that you may want to use to fit your objects:

- The Axis Aligned Bounding Box (AABB) is a box that aligns on the local frame of the object. This may be useful in a lot of cases because cubic objects (crates, walls…) have often the faces aligned with the three axis of the local frame.
- The Oriented Bounding Box (OBB) is a box that aligns automatically to the vertices of the object using principal component analysis (PCA). The main orientations of the points are extracted and used to orient the box.
- The Outside Capsule is a capsule that is oriented like the OBB.
- The Inside Capsule is similar to the Outside Capsule but it is not strictly a bounding volume because the two extremities are made shorter.
- The Sphere, which is simply a sphere.

These shapes provide fast and robust intersection tests and should be used in most cases if possible.

We explain now some three more geometries that you may use in your simulation: planes, convex meshes and penetration maps.

Planes, convex meshes and penetration maps

We now present three more collision geometries that you may want to fit to your objects: planes, convex meshes and penetration maps (pmaps).

Planes are robust shapes that split the space in two parts. Any object behind the plane is considered colliding with it. The normal of the plane is visible in the editor through a unit vector and indicates the front part of the plane. If you want to modify a plane, click on the corresponding [edit geometry] and use the plane [flip normal] button. Planes can only be used on static objects. They can also not be rotated with the arcball widget, but this will not be necessary as they always fit very well to the selected vertices.

Convex mesh shapes are the best solution when a more accurate collision model of a complex 3D model is needed. To create a convex mesh, select the part of the object that you want to fit and click on the [add convex] button in the toolbar. The convex hull is computed automatically. The maximum number of polygons for a convex mesh is limited to 256. You will be notified in a message box if the resulting convex hull exceeds this limit.

If you want to create a collision geometry that fits an arbitrary triangle mesh, you can use the penetration maps. PMaps use a voxel representation of the collision geometry instead of the triangles that defines its surface. To create a pmap, click on an object and click on the [Add PMap] button in the toolbar. A pmap can be created only for the whole object because it uses the volume defined by the object and not the vertices. However, it may happen that a pmap behave in a strange manner when running the physical simulation. This is normal, especially when using them on complex concave shapes. Please see the performance hints at the end of this tutorial for more information.

The next tutorial will explain how to rotate a geometry with the arcball widget.

Modifying primitives with the arcball

In this part we present the arcball widget. Arcball is a very simple tool that allows you to rotate your rigid bodies.

It may happen that the automatic shape fitting fails or does not orient it as you wanted. You can still modify the orientation using the arcball widget. Select any of the three orthographic views and click on the [modify geometry] button corresponding to the geometry that you want to modify. Now hold the left alt key on the keyboard to make the arcball appears and draw an arc on the sphere with the left mouse button. Drawing the arc will make the sphere rotate and the same rotation will be immediately applied to the selected geometry. You can also reset the rotation by clicking with the right mouse button. Release the alt key when finished. You can also zoom in (mouse wheel) to apply a more precise rotation.

You may also want to apply a constrained rotation for a simpler manipulation. You can apply a constrained rotation by clicking and dragging the mouse outside of the projected sphere. This can be done in any viewport to simulate a rotation around the x,y and z axis of the selected object.

Translating a geometry is done by holding the left ctrl key on the keyboard and dragging with the mouse left button. The geometry will exactly follow the cursor of the mouse at any zoom. Release the left mouse button and the ctrl key when finished.

Now that you can create and modify your collision geometries, we briefly present in the next tutorial the copy-pasting operation.

Copy-pasting

Copy-pasting allows you to copy the haptic properties of an object on an other object. This is usually performed when copies of objects are placed in the virtual environment. First, give some haptic properties to one of these object. Once finished, copy the properties by pressing CTRL+C on the keyboard. Then, select a second object and paste by pressing CTRL+V on the keyboard. All properties have been duplicated, you can even delete the haptic data of the first object if needed.

You are now able to create and modify haptic properties of the objects. The next part shows how to tune the simulation by introducing material properties.

Adding materials

Materials contains three values: restitution, static friction and dynamic friction. These values can be modified separately for each collision geometry. To access the material properties of a geometry, click on the corresponding [modify geometry] button in the toolbar. A new window will appear at the very bottom of the toolbar. You can modify the values by sliding the scrollbars or clicking on predefined materials. The following materials are already available: wood, marble, brick, steel, aluminium, glass, carpet, ice and magic ball. Once that you are satisfied with the parameters, click on the [ok] button in the toolbar to return to the previous state.

You are now ready to add complex haptic information to any object in the scene using the Haptic Scene Creator. The next tutorial explains how to run the simulation.

Immediate simulation of the system

This tutorial simply explains how to run the physical simulation of the system.

The simulation mode of the Haptic Scene Creator is immediate. This means that you can at any time run and observe the behaviour of the objects in real-time. To start the simulation, simply click on the [simulate] button in the toolbar. The perspective viewport is then displayed in fullscreen. You can also hide the toolbar by pressing the i key on the keyboard. You can still navigate as usual. You can even send some balls in the environment by pressing the spacebar on the keyboard. To return to the editor, click on simulate again.

You are now able to run the simulation dynamically. We finally discuss some performance hints in the next tutorial.

Performance hints and conclusion

This last tutorial deals with some performance hints that are especially important in large environments.

The computations done by a physics library are heavy, especially when dealing with complex scenes and/or complex objects. If possible, just use boxes, capsules and spheres. Box and capsules collisions are highly optimized and robust. If you want to use a collision object that is not a box, but not concave, a convex mesh will perfectly match. If your object is concave, decompose it into convex parts. According to the Ageia$^{\text{TM}}$ documentation, penetration maps are legacy objects and should not be used any more.

In building collision shapes for your objects, always remember that simpler is better and faster. Only resort to more complicated means when absolutely necessary.

If possible, the mass of the bodies should also be set to a value of 1.0. There is no predefined unit, just like in OpenGL, but the accuracy may suffer otherwise due to floating point numbers arithmetics. If you notice strange behavior with very light or heavy objects (actors jumps, vibrates, does not collide properly), you may try to set the mass near 1 and see if it solves the problem.

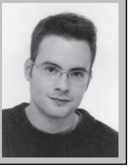
Concerning the materials, be aware that the numeric simulation also implies some errors. For example, a sphere that bounces on an horizontal plane may bounce higher than its original position if the restitutions of both colliding shapes are set to 1.

This tutorial is now finished. Congratulations, you are able to add haptic information to your virtual environment. You can always save and load the data, modify it, add more collision geometries to the objects and make them static or dynamic. You are also able to set materials to tune the surface properties of the shapes and you can run the simulation and observe the behaviour in real-time.

Renaud Ott

✉ : Cité-Devant 6
CH-1005 Lausanne
☎ : (+41) 78 741 30 27
📧 : renaud.ott@gmail.com

29 years old
Single
Swiss and French citizenship



Objective Statement: To put in practice my wide theoretical and practical knowledge of computer sciences by joining an innovative company with a promising future.

Education

Lausanne, Switzerland
2004 –2009
PhD in Computer Sciences at the Virtual Reality Laboratory in EPFL. Topic: "*Two-Handed Haptic Feedback in Generic Virtual Environments*".

Lyon, France – Lausanne
2002 – 2004
Master of Computer Sciences and Computer Graphics, with Honors at Claude Bernard University in Lyon, France.

Grenoble, France
2001-2002
Bachelor of Computer Sciences, with Honors at Joseph Fourrier University in Grenoble, France.

Grenoble, France
1999 – 2001
Technical Engineer Degree, with Honors in Computer Science at Technological Institute of Pierre Mendès-France University in Grenoble.

Annecy, France, 1998
Scientific Baccalauréat in Lycée Berthollet, Annecy, France.

Professional Experience

Lausanne, Switzerland
2004 – 2009
System Administrator of the VRlab – EPFL. Administration of more than 40 Windows/Linux computers, including 3 Windows Server 2003 servers, and 3 GNU/Linux Debian server (web, mail, intranet, print and file server, Active directory, etc.)

Grenoble
February 2003 to
July 2003
Programmer in the Leibniz Laboratory of IMAG, Grenoble. http://www-leibniz.imag.fr. Creation of a C++ program solving and optimizing a mathematical Great Graph Problem.

Paris
May 2002 to
July 2002
Programmer at MEDIADOC, the French leader of software for public libraries. http://www.mediadoc.com. Development of application intended to deploy and backup DVD/CD-ROM on a network.

Grenoble, France
June 2001 to
August 2001
Programmer in the ICA Laboratory of INPG, Grenoble. http://www-acroe.imag.fr. Creation of a 3D Graphical Rendering system of a 3D physical animation software: Mimesis.

Other

Academic and Research
8 **publications** in International Conferences Proceedings.
3 **articles** in International Journals.

Languages
French: Mother tongue, **English**: Fluent oral and written, **German**: Intermediate oral and written level, but lack of recent practice.

Computer Skills
Programming Language: C, C++, C#, Java, ADA, Prolog.
Libraries: NET Framework, JDK, OpenGL, OpenMP.
Web: HTML, CSS, PHP, JavaScript.
Systems: Administration and programming of Windows and Linux OS.
Database: SQL Language and administration of common databases.
Office Applications: MS Office, MS Exchange, LateX.

Hobbies and Interest
Sport: Ski, Tennis, Badminton, Squash, Swimming and Scuba Diving.
Interests: Science and music (as listener)
Hobbies: Remote Control Car Competition. Painting. Juggling.