transnationale
UNIVERSITEIT LIMBURG

School voor Informatietechnologie
Kennistechnologie, Informatica, Wiskunde, ICT

Haptic Feedback in Virtual Environments: Towards a Multi-modal Interface

Proefschrift voorgelegd tot het behalen van de graad van
Doctor in de Wetenschappen, richting Informatica,
te verdedigen door

Chris RAYMAEKERS

Promotor : Prof. dr. F. Van Reeth
Co-promotor : Prof. dr. K. Coninx

2002

LIMBURGS UNIVERSITAIR CENTRUM
PARTNER IN DE UNIVERSITEIT LIMBURG

Dankwoord

De realisatie van deze thesis zou onmogelijk zijn geweest zonder de hulp en steun die ik de afgelopen jaren heb gekregen.

Ik zou in de eerste plaats mijn promoter prof. dr. Frank Van Reeth en co-promotor prof. dr. Karin Coninx willen bedanken. Dankzij hen heb ik de boeiende wereld van virtual environments en human-computer interaction ontdekt. Ik heb veel plezier beleefd aan onze discussies over het onderzoek. Hun raad en steun waren zeer belangrijk voor mij.

Ik wil mijn collega's bedanken voor de hulp die ik heb gehad bij het maken van mijn thesis en voor de vriendschap die ik de afgelopen jaren van hen heb gekregen. Hierbij zou ik in het bijzonder Joan De Boeck, Tom De Weyer, Fabian Di Fiore, Gert Vansichem en Tom Van Laerhoven willen bedanken. Ik hoop dat we nog lang op deze manier kunnen blijven samenwerken.

Ik zou ook prof. dr. Eddy Flerackers, directeur van EDM, willen bedanken voor de kansen die hij mij gegeven heeft. Ik ben blij dat ik in de stimulerende omgeving van dit onderzoeksinstituut mijn onderzoek heb kunnen uitvoeren.

Tenslotte zou ik mijn echtgenote, An, mijn ouders en mijn familie willen bedanken voor de steun die zij mij hebben gegeven. Zij hebben deze ervaring de moeite waard gemaakt.

Abstract

The goal of virtual environments research is to give the user an experience that is indistinguishable from the real world. Nevertheless, research has mostly concentrated on visual output. Our research has shown that visual feedback limits the realism of the user's experience. In order to allow for an intuitive interaction with the virtual environment, the user interface must be multi-modal. In virtual environments, touch is a useful modality, since people rely on touch in real world interactions. Furthermore, devices that can generate haptic feedback became recently available.

The research described in this thesis incorporates haptic feedback in virtual environments, thus extending the possible interactions in these environments. First, the use of touch was assessed using passive touch in the domain of cut-out animations and active touch in the domain of curve drawing. Next, the modelling environment, we use in our research, was enhanced with haptic feedback and new interaction techniques were developed. Furthermore, haptic rendering techniques were developed in order to make 3D objects touchable.

Since user interfaces cannot be analytically evaluated, two user experiments were conducted in order to evaluate our interaction techniques. The results prove our interaction techniques to be useful.

In haptic environments, the user often needs both hands to manipulate devices. This means that a mouse and keyboard cannot be used to give commands. A natural solution is the use of speech to give commands. Indeed, we have shown that speech is a useful modality in a haptic environments.

We believe that the use of haptic feedback is an important step towards the creation of an intuitive and flexible multi-modal user interface for virtual environments.

Contents

Chapter 1

Introduction

Virtual environments have proven to be beneficial for a number of applications, including training applications, computer aided design (CAD) and entertainment. The benefits of these environments are only be useful if attention is paid to the user. This thesis treats the research area that combines virtual environments and human-computer interaction. The starting point of this research is interaction for 3D object modelling applications. Since modelling applications have a large user involvement, they require sophisticated interaction. We thus believe modelling applications are a good test case for human-computer interface techniques for virtual environments.

Traditional virtual environment only provide visual feedback to the user. This limits the interaction possibilities in this world and makes the experience less intuitive to the user. People constantly use multiple modalities when interaction with the real world, we use our sight, touch, smell and hearing in order to receive information about our surroundings. A virtual environment that does not make use of multiple modalities therefore limits its users.

Our work concentrates on multi-modal interfaces for virtual environments. We elaborates on the current knowledge about human-computer interaction and virtual environments, more specifically on hybrid 2D/3D interaction in virtual environments. A solution for the problems that arise in this kind of interface is sought in multi-modal interaction.

Since our sense of touch is very important in our daily experiences and since devices that make use of our sense of touch (called haptic devices) recently became available, our research concentrates on haptic feedback in virtual environments. More precisely, since people use touch when manipulating objects, we especially look at haptic feedback in manipulation of virtual environments.

The impact of interaction techniques on the users can only be determined by the users themselves. We therefore conducted two user experiments in order to assess the usefulness of our interaction techniques.

In chapter 2, virtual environments and manipulation techniques in virtual environments are introduced. Furthermore, our 3D object modelling environment, ICOME, is introduced. The problems encountered in this project are used as a basis to motivate multi-modal user interfaces, which are explained at the end of this chapter.

Before the use of active touch can be explained, chapter 3 elaborates on passive touch. The history of passive touch, along with various definitions, are given. Some representative examples of passive touch applications are explained in more detail. Finally, our passive touch application for controlling cut-out animations is expounded.

Active touch is discussed starting from chapter 4. This chapter introduces the terminology of active touch and its relationship with human perception. Various devices that support active touch are discussed; one particular device, the PHANToM device, is discussed in more detail since this the device we use in our research. The use of force feedback in WIMP interface is discussed as a case study of active feedback along with our application for using haptic feedback for curve drawing.

The remainder of this thesis concentrates on virtual environments. Chapter 5 discusses haptic feedback can be used in virtual environments by giving examples of a number of application areas. Attention is paid to haptic rendering techniques that are used to make a virtual environment touchable. We introduce haptic rendering techniques for two alternative object representations. Finally, this chapter elaborates on the integration of haptic into ICOME, the modelling environment that was introduced in chapter 2 and on two cases of user interaction that are supported in this improved framework.

Chapter 6 concentrates on the evaluation of interaction techniques. It explains how a user experiment should be conducted and the particular points of interest with user experiments. Special attention is paid to the statistical analysis of the results. In order to asses their validity, the interaction techniques of chapter 6 were evaluated in formal user experiments. These are discussed at the end of this chapter.

Another modality, speech input is the subject of chapter 7. This chapter first discusses speech interfaces in general and speech interfaces for virtual environments. Our proof-of-concept application of a speech interface for haptic environments in presented.

Finally, chapter 8 gives some directions for future work in the domains of haptic research and multi-modal interfaces. In chapter 9 summarises the realisations of our research.

Chapter 2

Interfaces for Virtual Environments

Contents

2.1 Introduction

This thesis treats multi-modal interfaces for virtual environments. In order to understand the underlying issues of virtual environments it is important to understand what virtual environments imply.

No exact definition of virtual environments can be given, since a lot of different disciplines are involved in virtual environments. We can however state that over the past few years the term "virtual environment" has replaced the term "virtual reality". Whereas virtual reality is a contradiction, the term virtual environments emphasises that we deal with worlds that are artificially created.

Although a lot of different definitions can be found in literature, the following definitions treat the aspects that are important for this thesis.

> "Virtual reality (VR) refers to the computer-generated simulation of a world, or a subset of it, in which the user is immersed. It represents the state of the art in multimedia systems but concentrates on the visual senses." (Dix et al., 1998)

> "Virtual reality is the use of various computer graphics systems in combination with various display and interface devices to provide the effect of immersion in an interactive three-dimensional computer-generated environment in which the virtual objects have spatial presence. We call this interactive three-dimensional computer-generated environment a virtual environment." (Bryson, 1994)

> "Virtual environments are synthetic sensory experiences that communicate physical and abstract components to a human operator or participant. The synthetic sensory experience is generated by a computer system that one day may present an interface to the human sensory systems that is indistinguishable from the real physical world." (Kalawski, 1993)

These definitions concentrate on the importance of the sensory system. Dix et al. indicate that at this moment most work concentrates on the visual system. However, Kalawski indicates that the goal of virtual environments is to present an interface that is realistic. In order to achieve this goal, user interface for virtual environments should be multi-modal: a visual model that cannot be touched is not realistic for the user.

Although the study of virtual environments depends a lot on technical advances — both in hardware and in software — the human component of the virtual environments is very important. Without a good understanding of the human factor, it is not possible to build a good and intuitive user interface for virtual environments:

> "The careful consideration of human factors represents a fundamental dimension of research and development in virtual reality (VR) and virtual environments (VEs). There is a diverse range of human factors: some of them have been well studied and some remain to be fully understood and are the focus of much ongoing research." (Chen et al., 1998)

The remainder of this chapter will discuss different manipulation techniques that are currently used in virtual environments. Next, a number of virtual environments will be discussed in order to present different application areas of virtual environments and to present the usage of these different manipulation techniques.

These manipulation techniques and example environments are a base for our research into an object modelling environment: ICOME. The system architecture and the interactions that it allows will be discussed. From this discussion, together with examples from literature, we will show that multimodality is necessary in order to come to a good user interface.

2.2 Manipulation Techniques

One of the most important aspects of a virtual environment is the interaction style for manipulating objects, because it influences the intuitivity of the interface. This section will therefore discuss different manipulation techniques that are being used in virtual environments.

2.2.1 Classification

Different interaction techniques are used in virtual environments. However similarities between these interaction techniques can be de distinguished. Poupyrev et al. (1998) identified three classification categories:

- Exocentric metaphors
- Egocentric metaphors
  - Virtual Pointer metaphors
  - Virtual Hand metaphors

They evaluated different egocentric manipulation metaphors and came to the conclusion that there is no “best” manipulation metaphor: which metaphor should be used depends on the situation and context.

We will next discuss examples of each metaphor category.

2.2.2 Exocentric metaphor

Worlds in Miniature (WIM) is a exocentric manipulation metaphor (Stoakley et al., 1995). This means that the user has an outside view of the virtual environment and is not immersed in (this representation of) this virtual environment.

WIM presents more than a manipulation metaphor. In fact, it is also a navigation metaphor. In WIM, the user views the virtual environment through a head-mounted device. A miniature of the virtual environment is also presented to the user, thus allowing the user to view objects (in miniature) which would otherwise be occluded. All manipulations on the WIM are reflected on the virtual world. The WIM interface also allows the user to change the view point. This change is not reflected immediately because this is very disturbing. Instead, after the view point has been changed in the WIM, the according change of view in the virtual environment is animated.

The WIM is virtually attached to a physical clipboard, which is magnetically tracked. This allows the user to move the WIM in the virtual environment. Object manipulation are performed using a physical prop (Hinckley et al., 1994a, see also section 3.2.1); in this case a magnetically tracked tennis ball equipped with two buttons.

From reactions of users, the authors concluded that the WIM metaphor is intuitive to use and is general applicable across a wide range of applications.

2.2.3 Virtual pointer metaphor

With an egocentric manipulation technique, the user’s viewpoint is located inside the virtual environment. The CAD program JDCAD (Liang and Green, 1994) and its derivative JDCAD+ (Green and Halliday, 1996) make use of egocentric manipulation techniques, more specifically a virtual pointer metaphor. The user operates a bat, a 6DOF input device, which is mechanically tracked. At the user’s hand, a spotlight is defined, which provides a conic selection volume. Objects that intersect this volume can be selected.

The bat was also used to operate a spherical menu (see figure 2.1). Again the cone is used to select menu items. Another menu item can be selected by turning the daisy menu. A problem with this techniques lies in the fact that the menu items have to be represented by 3D icons and that the menu does not scale well to a large number of menu items.

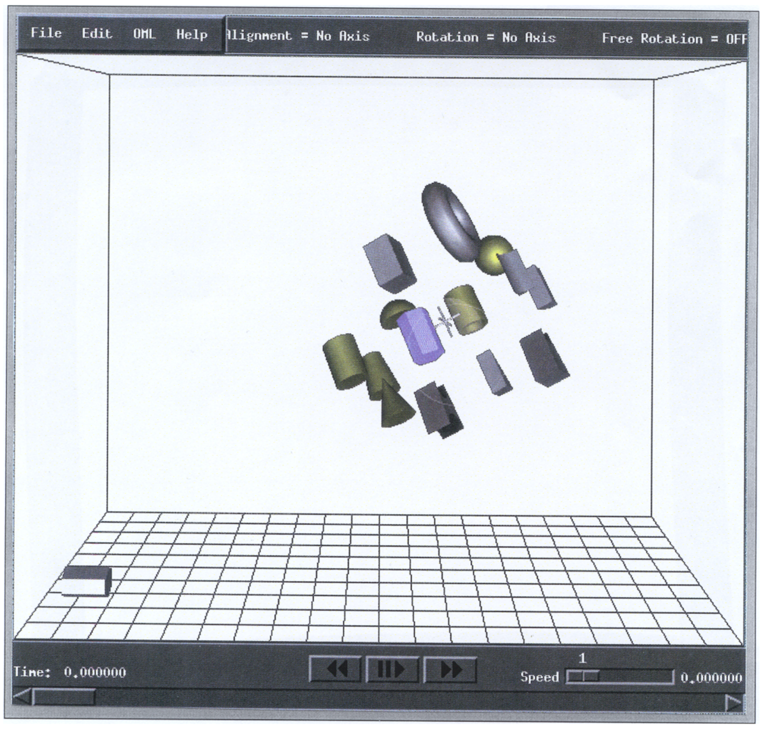


Figure 2.1: Spherical menu from JDCAD/JDCAD+

2.2.4 Virtual hand metaphor

A virtual hand metaphor provides the user with a virtual hand. The movement of the user's hand is mapped onto this virtual hand. An example of this metaphors is the Go-Go interaction technique (Poupyrev et al., 1996). Unlike other virtual hand techniques, the mapping of the user's hand is not linear. It is linear when the hand is close to the user, but the mapping is not linear when the user's hand is further away.

This both allows the user to make precise movements near his own position (the mapping is linear) and to reach for objects that are not located in the user's neighbourhood (the mapping is non-linear).

2.3 Examples of Virtual Environments

This section provides a number of virtual environments which represent the applications of current virtual environments.

2.3.1 CHIMP

The UNC-Chapel Hill Immersive Modeling Program (CHIMP) is a virtual reality application for architectural design (Mine, 1996). The CHIMP program allows architects to design rooms using existing objects.

The user is immersed in the virtual environment by means of a magnetically tracked head-mounted display. As input devices, two bats are used, one for each hand.

Similar to JDCAD, a spotlight metaphor is used for object selection. However, since rotations of an object are difficult to achieve using a virtual pointer, follow-up research of CHIMP uses a virtual hand together with automatic scaling of the world (Mine et al., 1997).

Menu selection is provided by a "look-at menu": when the user presses the menu button on his bat, a pop-up menu appears. By looking at a menu item and releasing the menu button, the menu item is selected.

Two-handed input is used for translating, rotating and scaling objects: the distance between the user's hands determines by how much the object should be translated, rotated or scaled respectively. This two-handed metaphor is also used to determine the navigation metaphor. It is possible to orbit a selected object in order to inspect it from all sides without having to change to orientation of the object; the distance between the user's hands determine the distance to the object. Its is also possible to fly through the virtual environment by pointing in the direction one wants to go. The CHIMP system also uses WIM (see section 2.2.2) to view the virtual environment.

Two-handed input is also used for control panel action. Whenever a control panel is needed in the interaction, it is attached to the user's non-dominant hand and can be manipulated by the dominant hand.

2.3.2 IM-Designer

Unlike CHIMP, IM-Designer(Coninx, 1997; Coninx et al., 1997) supports the creation of new objects starting from scratch, whereas in CHIMP only existing objects can be used.

A BOOM (Binocular Omni-Orientational Monitor, see figure 2.2) is used to immerse the user in the virtual environment. Input is provided with a pinch glove (the non-dominant hand is used to operate the BOOM).

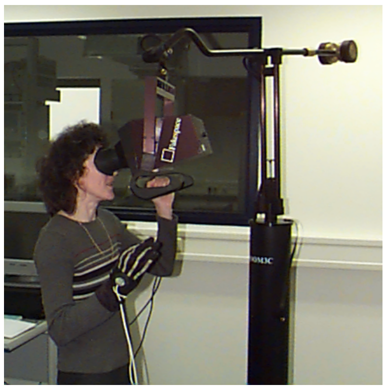


Figure 2.2: Interaction with BOOM and pinch glove

IM-Designer makes use of a “hybrid 2D/3D” user interface, as depicted in figure 2.3: all object manipulations that can be executed in 3D are performed in 3D. These manipulations include selecting and moving objects. Some other operations require the use of menu’s and dialogs. These menus and dialogs are presented as 2D widgets that float in 3D space, hence the name hybrid 2D/3D interface. An example of such a 2D widget is a menu which allows the user to choose a shape that has to be created in the virtual world.

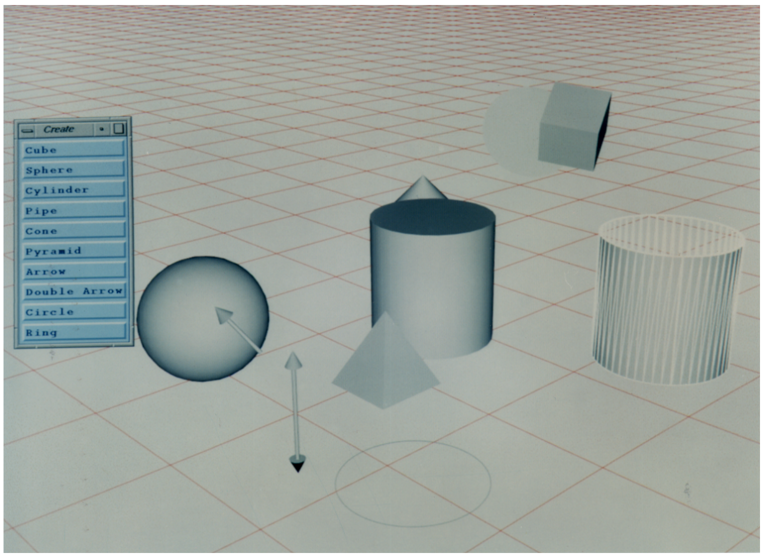
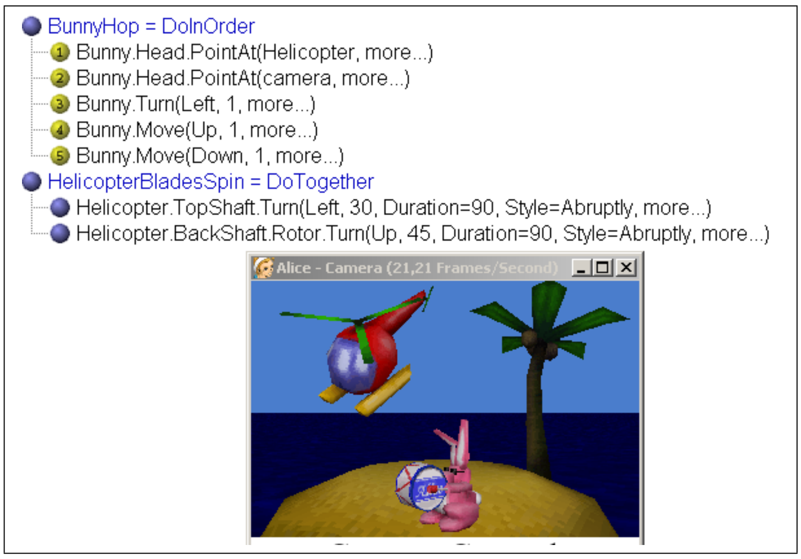


Figure 2.3: Hybrid 2D/3D interface in IM-Designer

2.3.3 Alice

Alice (Conway et al., 2000) is an authoring tool for interactive 3D graphics. This program is aimed at a large and diverse audience and therefore uses a through-the-window metaphor with standard PC hardware.

Because Alice describes time-based and interactive behaviour of 3D objects, its basic elements are objects and actions on the objects. Since no 3D input devices are used, these actions can be defined using a 2D GUI which creates Phyton scripts that operate on the objects (see figure 2.4).



```
● BunnyHop = DoInOrder
    1 Bunny.Head.PointAt(Helicopter, more...)
    2 Bunny.Head.PointAt(camera, more...)
    3 Bunny.Turn(Left, 1, more...)
    4 Bunny.Move(Up, 1, more...)
    5 Bunny.Move(Down, 1, more...)
● HelicopterBladesSpin = DoTogether
    ● Helicopter.TopShaft.Turn(Left, 30, Duration=90, Style=Abruptly, more...)
    ● Helicopter.BackShaft.Rotor.Turn(Up, 45, Duration=90, Style=Abruptly, more...)
```

Figure 2.4: Example Alice scene

The GUI is developed in such a manner that non-technical people can work with the program: all technical terms are replaced by common words. For instance, objects do not move in a x, y, z direction, but move forward or upward. The program also uses direct manipulation on the objects and actions, but 3D manipulations with a 2D mouse are not always obvious.

2.3.4 TypoToons

TypoToons (Flerackers et al., 2000, 2001) is an interactive television series, which uses networked personal computers to allow children to perform tasks together in a virtual environment.

In this television series, four children have to perform a number of tasks in order to win letters. With these letters, they can make a word. The actions of the users are recorded together with camera recordings. The director can later use real and virtual images during a broadcast. An example broadcast image is shown in figure 2.5.

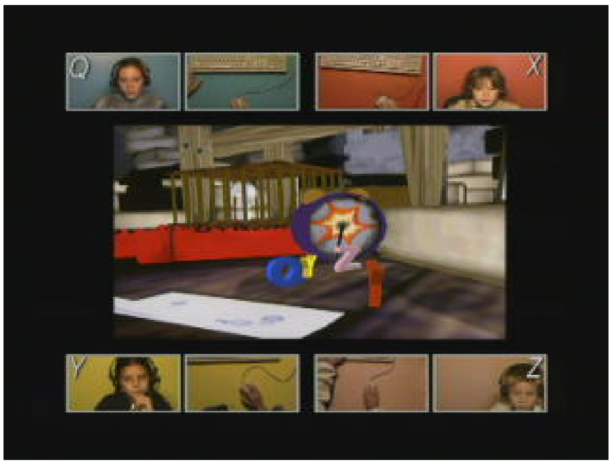


Figure 2.5: TypoToons image that was broadcasted

The main interaction of TypoToons consist of the movement of the children's avatars (which are depicted by letters). Further attention has been spent on smooth animation on standard PC hardware and the collaboration over a network.

2.3.5 PSDoom

The process manager PSDoom (Chao, 2001) is an UNIX administration tool that is based on the first-person shooter game "Doom". It provides a virtual environment in which the processes running on the computer are represented by monsters.

The user can lower the priority of a process by wounding the corresponding monster. If the monster is killed, the process is also killed. On the other hand, processes can defend themselves by they attacking the user.

Although PSDoom is at first sight not a useful application, this kind of system administration can has its merits, since on highly used system, random killing of processes is frequently used to lower the number of processes. Since processes can also attack each other, a system with a lot of processes will have more kills because each monster has more enemies. A condition is that important processes are stronger (they are more difficult to wound or kill) and that they are bigger (they can easily be identified by the user).

This program defines another application area for virtual environments: performing system adminstration by representing abstract resources, such as processes, as characters within a virtual environment.

2.4 ICOME

ICOME is an abbreviation of Immersive Collaborative 3D Object Modelling Environment (Raymaekers et al., 1999). 3D Object modelling deals with defining the geometry of the objects in computer generated scenes. The scenes find their application in a number of situations, as there are virtual worlds in Virtual Reality software, but also animation films and simulations.

Section 2.3.2 described the immersive modelling and hybrid 2D/3D interaction of IM-Designer. Our research project ICOME is based on that experience. The original concept of the research with regard to immersive modelling has been extended in several ways. The main improvement of the concept is the attention that is paid to collaborative modelling. IM-Designer used to be a single user program, but we realised that collaboration would open a whole new perspective to 3D-modelling. Although computer support for collaborative work in a design context is emerging, we are currently not aware of computer supported collaborative 3D object design.

ICOME also differs from its predecessor with regard to the user interface paradigm. In addition to immersive modelling, the project now supports non-immersive interaction with the system. The non-immersive user interface can be used to overcome restrictions of immersive working (e.g. precise manipulations), but also due to practical reasons such as the available hardware and software. The possible co-existence of immersive and non-immersive clients influences the design of the user interface and the choice of interaction devices. The system aims to provide similar interaction in the immersive and non-immersive situation. Besides addressing collaboration issues and user interface design, ICOME is a platform to experiment with immersive modelling features and with the design of a suitable object hierarchy to represent the virtual scenes.

2.4.1 System architecture

The system architecture of ICOME is based on a client/server structure (as depicted in figure 2.6). The server stores all information concerning the virtual world being modelled and is responsible for the synchronisation of the clients. The client provides the user interfaces for the collaborative modelling system. Client software depends on the computer being used as a modelling workstation, and the interaction devices used for modelling tasks.

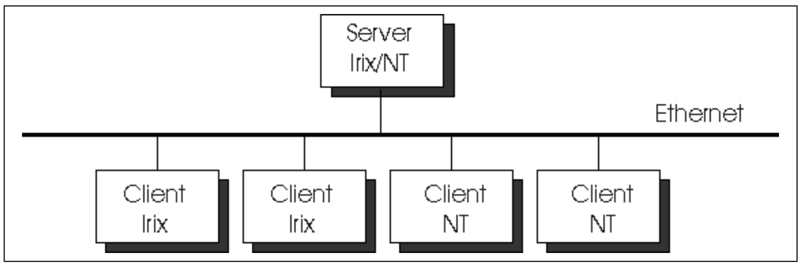


Figure 2.6: Client-server model

Server

The server is responsible for information storage and for synchronisation issues following from collaborative use of the modelling system. Therefore the server addresses collaboration issues such as floor control, and inhibits several designers to modify an object at the same time.

All information about the virtual world being modelled (also called the design hereafter) is kept in a database on the server. The section about the client architecture elaborates on the object hierarchy that is used to represent the designs.

The object hierarchy corresponding to the design is stored on the server, but it is duplicated on each client that connects to the modelling system. When a new client joins a collaborative modelling session, it receives its working copy of the design. The server is responsible for the management of the object hierarchy and for the synchronisation of the client's editing operations. When one of the clients modifies part of the design, all other clients must be informed about this change. The client performing the change notifies the server, which in turn notifies the other clients. Notifications are sent using programmer-defined messages between the clients and the server. We distinguish between four kinds of notifications, depending on the sender and receiver(s) of the messages:

- A first category are notifications sent by the server to all clients. An example would be the message COLOBJ_CREATE, which the server sends to obligate all clients to make a new object.
- A second category of notifications is sent to all clients but the client that notified the server. When one of the clients interactively moves an

object, the server sends the COLOBJ_TRANSLATE message to the other clients to update the object position in their user interfaces.

- In some situations the server only has to acknowledge the notification of the client, e.g. the message SELECT_COLOBJ confirms the selection of an object by the client.
- A last category of notifications only flows from client to server and does not result in a reply or forwarding to other clients. The message DESELECT_COLOBJ, which a client sends to notify the server that it releases a requested object, belongs to this category.

Using this notification system, the server maintains the consistency between the design information that is visible on all clients. Data reservation in the collaborative environment is another aspect of synchronisation that belongs to the server's responsibilities. In order to preserve the consistency of the object hierarchy, part of the hierarchy is allocated to a client requesting a modification. The strategy used by a collaborative system to handle requests such as data reservation is commonly called floor control. Although each client has its own working copy of the object hierarchy, conceptually the database on the server is a shared resource that can be locked. Before a client can modify an item (e.g. an object or a material), that item has to be requested from the server. A request for a material will simply be granted or denied. When evaluating a request for an object, the server also has to consider the status of its parent object and child objects. An object can not be selected when another user already has selected the parent (or ancestor) object. Nor is it possible to select an object if a child (descendant) object of this object is already selected. On the other hand, siblings and descendants of siblings can simultaneously be selected. In summary, a request for an object is only granted by the server if no other object is selected, or if one or more objects in a separate subtree of the object hierarchy are selected.

We can conclude from informal user tests with ICOME that the described floor control strategy does not obstruct constructive collaborative work in the modelling system.

Client

It has been pointed out above that the ICOME modelling system supports immersive as well as non-immersive clients at the same time. The user interface of the client provides the designer with the functionality to creatively work with the object hierarchy representing a design. Besides choosing for immersive or non-immersive interaction with the system, there is the option to use a SGI IRIX client or a Windows NT PC client. Figure 2.6 shows some combinations of operating system.

In general, the client software can be represented as an architecture consisting of three basic blocks (see figure 2.7):

1. the object hierarchy
2. the immersive or non-immersive user interface and
3. the part of the client software that is responsible for communication over the network.

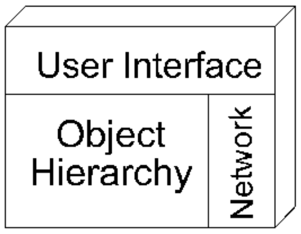


Figure 2.7: Client Architecture

The object hierarchy block deals with the object representation in the object hierarchy and with rendering issues. The section about the server has described how the server and the clients exploit message-based communication to ensure efficient updating of the object hierarchy. On the source code level, each kind of client is based on the same C++ classes as representation for the object hierarchy. Changes in the object hierarchy are thus reflected in each kind of client after recompilation on both platforms (SGI IRIX and Windows NT).

Clients on both platforms use the OpenGL library for rendering the objects. By placing all OpenGL function calls in the object hierarchy structures, we were able to ensure that all three clients represent the object hierarchy in the same way.

The user interface block in the client architecture controls the viewing and interaction devices, and supports the user with modelling tools to edit the design. The user interface makes use of native interfacing libraries for the different operating systems.

The network layer differs between IRIX and Windows NT code, although in both cases the same support is provided for the client's user interface. Differences have to do with memory management and sockets in the respective operating systems. IRIX (and Unices in general) uses big endian (high-order bit left) to store variables in main memory, while Windows NT uses little endian (high-order bit right). The server also uses big endian for network communications, because this is a highly used standard. This means that the Windows NT client has to change the variables' endian when sending and receiving data. On the IRIX platform Berkeley sockets are used, while WinSock version 2 is used on NT clients. The issues with regard to memory management and sockets, combined with performance considerations, justify our decision to develop two different network layers optimally exploiting features of the operating system they are developed for. As a consequence of having several client types, each of them uses the same kind of data representation when communicating with the server. The server from its side can always operate in the same way, without having to compensate for differences in various implementations of the client. This improves the overall performance of the server, because it avoids a series of computations and therefore more request messages can be served in the same time span.

As a final remark in the context of the client architecture, we would like to emphasise that the clients are also capable of running without the network mode. This enables the user to prepare a model without having to be connected to a server, and to integrate it later into the server's object database.

Object hierarchy

The term object hierarchy is used to describe the hierarchical relations between the virtual objects of which a scene (or virtual world) is composed (see figure 2.8). Each object may have child objects, which may in turn have child objects. Each object has its own co-ordinate system relative to its parent's co-ordinate system. The object references a set of vertices, edges, triangles and patches (3D surfaces). Object-orientation is exploited in the object hierarchy in order to allow easy extension and addition of new functionality.

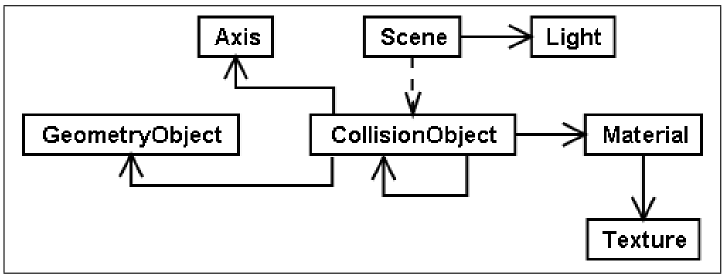


Figure 2.8: Object hierarchy

At the root of the object hierarchy is the Scene class, which represents the overall virtual world (or "design" in the modelling context). Scene has a co-ordinate system consisting of translation, rotation and scaling parameters. These parameters are stored with respect to the co-ordinate system of OpenGL, allowing completed scenes to be combined in one scene. The Scene object references a list of all parent objects and all objects, and the materials and textures used in the virtual environment. It also refers to data concerning the lights that are used in the virtual world.

Every object in the virtual world is an instance of the CollisionObject class, and contains a material object and a GeometryObject (vertices, triangles) and/or a list of child objects which are CollisionObject instances in their turn. It is important to mention that a CollisionObject instance must contain either a GeometryObject or a list of child objects to be useful. CollisionObject instances are used to perform collision detection.

Separating the GeometryObject from the other attributes is favourable for copy and clone operations. Copy means duplicating the complete instance variables of the object, while cloning stands for creating additional references to existing objects. Currently, six copy/clone operations are supported:

1. copy the object
2. copy the object and its children
3. clone the object
4. clone the object and its children
5. copy the object and clone its children
6. clone the object and copy its children.

Materials are always cloned. Consider the following example to illustrate the usability of the copy and clone operations. Suppose two similar tables have to be designed. When designing the first table, one leg is modelled and cloned to realise the remaining three legs. As a result, changing the colour of one leg causes the other legs to be adjusted too. The second table is created using the "copy the object and its children" operation. This means that the legs of the second table are clones from each other, but not from the legs of the first table. With other words, adjusting the shape of a leg of the second table causes all legs of the second table to be changed without affecting the legs of the first table. This interaction is depicted in figure 2.9.

The Vertex class describes a point in the co-ordinate system of the object, with pointers to the edges and triangles that are connected with that vertex. Each Vertex object is represented relatively to the objects co-ordinate system. The vertices are invariant when moving an object, thus only the co-ordinate system of the object has to be adjusted.

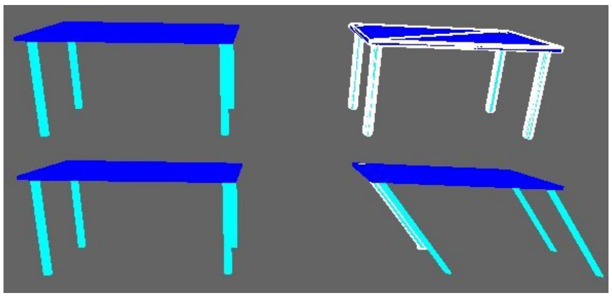


Figure 2.9: Example of cloning

The Triangle class describes pointers to its three defining vertices, pointers to its edges and a pointer to the material that is being defined on the scene level, so that different objects can reference the same material objects. The texture co-ordinates are also kept in Triangle (these are the U, V co-ordinates). This is done because one vertex could be a member of different triangles with different textures, so that one vertex could have different texture co-ordinates for the different triangles.

The Material class contains the RGB values of the triangles and the material specifications of the triangles such as ambient, diffuse, specular, shininess and emission. A Material object may also contain a pointer to texture information.

Reference counting is used to save the amount of allocated memory. A vertex typically lies in the cross-section of several triangles. The triangles reference one and the same vertex object. The Vertex class holds a reference count data member, keeping track of the number of triangles referencing this particular vertex. Before deleting a triangle, the algorithm will check the reference count to determine if that triangle is the only one that uses the vertices. If so, the vertices will be deleted. Otherwise the reference count is decremented. Reference counting is also used in other situations, such as to manage the use of textures in the scene.

An hierarchic object structure has many advantages. For example, the performance of collision detection is better because the algorithm has to be performed only on the top level with the objects that are connected directly to the scene. Object hierarchies are applied successfully in the domain of computer animation. Similar benefits are found in the context of virtual reality, where performance considerations are important to enable real-time updating of the virtual world. Therefore, our interactive immersive modelling environment ICOME creates objects and scenes according to an hierarchical structure, that is beneficial for the virtual environment in which the scene will be used. Typical examples are child objects that move together with their parent object, and easy specification of animation because every object has its own co-ordinate system. For instance, the user can create a car, where the car tires are defined as child objects of the car body. If the user specifies an animation sequence with a driving car, the child objects are informed about the parent's movement so that the child objects adjusts their co-ordinate system accordingly. If the car moves forward, then the tires turn in the correct direction. Without the hierarchical structure all the vertices of the car tires would have to be recalculated.

The VRML 2 file format is used to store ICOME designs, which allows porting from designs from or to other platforms using the VRML standard.

2.4.2 Interaction

The usability of the modelling system depends on the realisation of uniform interaction for the immersive and non-immersive clients. Designers must be able to migrate from immersive to non-immersive clients and vice versa, while preserving a similar interaction (as explained in (Coninx et al., 1999)). Although we are convinced that immersion has added value for a number of design situations, we are committed to design the interaction in a way that has benefits for users on non-immersive clients too.

We will concentrate here on information with regard to interaction. The interaction paradigm has been inspired on the hybrid 2D/3D interaction in IM-Designer. The general idea is that 2D and 3D interaction are combined, and each of them is used in the situations it is suited for. Our conclusion is that 2D interaction is favourable for interfacing elements such as menus and dialog boxes. Furthermore, 3D interaction is intuitive for a selection of object modelling tasks such as object positioning, resizing, ...

Direct manipulation is the dominant interaction style for 2D as well as for 3D interaction. Though details of the interaction scenarios are adjusted depending on the client in case, the overall idea of direct manipulation is implemented on immersive as well as non-immersive clients. Much attention is given to direct manipulation for advanced modelling features such as extrusion, free-form deformation and texture mapping.

The immersive clients in our experiments are SGI Onyx computers. A BOOM (see figure 2.2) is used as the immersive display device. An advantage of using a BOOM as the output device is the intuitive interface it provides to navigate through the virtual world. Indeed, the BOOM is equipped with two programmable handles which are commonly used to define the direction of the movement (forward/backward) and the speed of the movement.

A Fakespace pinch glove is used as the interaction device for immersive clients as in figure 2.2. As opposed to a regular glove, the pinch glove does not measure bending of the fingers. A Polhemus tracker is mounted on the back of the pinch glove to measure its 3D position and orientation. The contact pads at the fingertips of the pinch glove can be pressed to generate interaction events (comparable to pressing a mouse button). Experimental use demonstrates that the pinch glove is a suitable interaction device in the context of immersive interaction with the hybrid user interface. It is used for the interaction with menus and dialogs as well as for the manipulation of virtual objects. The position of the glove, relative to the virtual camera, is shown by drawing a virtual hand.

SGI IRIX workstations as well as Windows NT PCs can be used as non-immersive clients. Obviously, immersive viewing is not provided in this set up. However, different clients strive for similar user interfaces. This means that the interaction is designed in such a way that moving from an immersive client to a non-immersive client or vice versa shouldn't confuse the user. Non-immersive clients can use a 3D mouse (see figure 2.10) as the input device. This device replaces both the pinch glove and the navigation buttons on the BOOM.

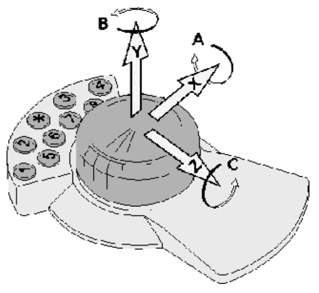


Figure 2.10: Magellan space mouse

By using one of the eight buttons of the 3D mouse, the events corresponding to pressing contact pads with the pinch glove can be simulated. Other mouse buttons can be used for repositioning the hand relative to the virtual camera or for repositioning the camera relative to the virtual world's origin. The 3D mouse is also used to toggle between camera mode and hand mode, where respectively the camera and the hand are moved according to the movement of the 3D mouse.

A non-immersive user interface is not as intuitively as the use of the BOOM combined with the pinch glove, but our experiences have shown that users are able to adapt to a 3D mouse in a relatively short time span. In our opinion a 3D mouse is also very good for accurate movement in a virtual 3D world (which is traditionally a weak point of direct 3D interaction devices that has to be overcome by placing constraints on the movement), although De Boeck et al. (2001) showed that navigation with the 3D mouse is only suitable for expert users. Obviously, the spatial awareness resulting from immersive viewing through the BOOM is an important advantage for the immersive experimental set-up.

2.4.3 Functionality

In ICOME, the designer starts from primitive objects and modelling tools to build objects from scratch. The objects can be combined into more complex objects and finally in virtual scenes. Immersive as well as non-immersive clients offer similar modelling features (also called modelling tools) for the creation and manipulation of 3D objects.

Object creation

In order to allow users to create objects, a number of primitives (e.g. triangle, square, circle, pyramid, cube, sphere...) are provided. These primitives can be used as a base for the 3D object that the user wants to create.

In addition to using primitive objects, new 3D objects can be created based on an existing 2D object by means of the extrude functionality. The extrude tool applies a series of rotations, scalings and translations on a 2D object in order to create a 3D object. A cone can be extruded by rotations from a triangle, or by translations and scaling of a circle. A torus is extruded from a circle by repeatedly applying translations and rotations on a circle.

Another useful tool is 3D extrude, which manipulates 3D objects. For instance to create stairs. The object is created in four major steps (see figure 2.11):

1. create a square with a wood texture
2. extrude the square to create a step of the stairs
3. select the single step, select the rotation point, and move a copy of the step to the correct position
4. repeat the previous step as many times as desired (this step is automated)

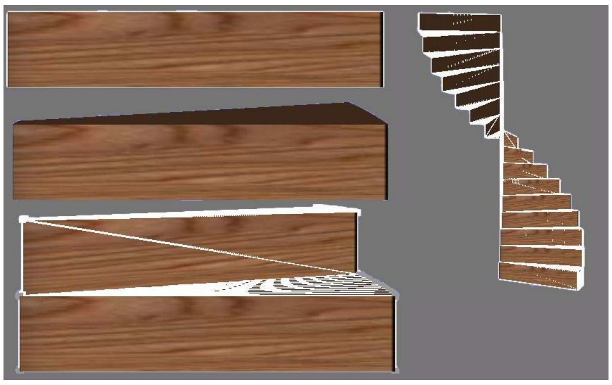


Figure 2.11: Example of object creation: stairs

Basic manipulation

A first manipulation tool that is provided is copying and deleting individual objects. By performing these operations on its child objects, the object is changed in an indirect way. Another way to manipulate objects is by dragging individual vertices to another location in the 3D world as shown in figure 2.12

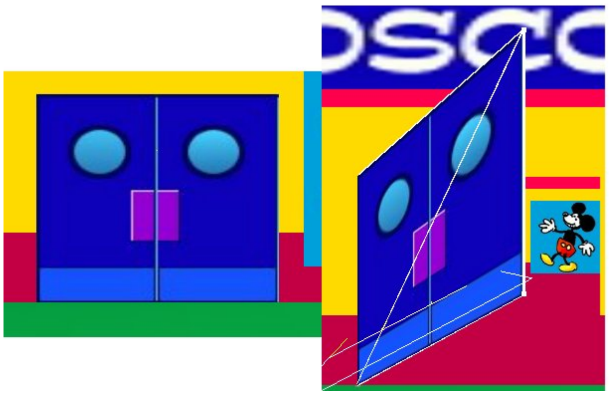


Figure 2.12: Dragging

Free-form deformation

Free-form deformation is a tool that can manipulate a polygon mesh. When the user wants to deform such a mesh, a bounding box is created around the object. A series of reference points is defined on the outside of the bounding box. Finally, each vertex in the object receives three co-ordinates that define the position of the vertex in terms of the reference points (Sederberg and Parry, 1986). The polygon mesh can be deformed by dragging the reference points.

Figure 2.13 gives an example of a free-form deformation. First a square is subdivided to form a polygon mesh. In order to accomplish this, a triangle subdivision tool is recursively applied. This tool subdivides a triangle in four new triangles. A free-form deformation is then applied on the polygon mesh in order to become the desired object (another example of the use of our algorithm can be found in (De Weyer et al., 1999)).

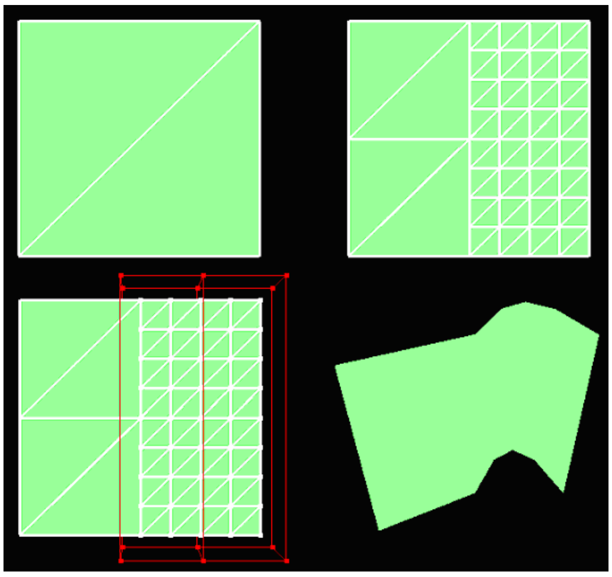


Figure 2.13: Free-from deformation

Navigation

Navigation through the environment is used to move the viewpoint on the object being modelled. This provides the designer with different views on the same object in order to get insight in the object geometry. It also enables the designer to reach remote objects before they can be manipulated.

The interaction scenarios to navigate in ICOME are based upon the input devices being used at the clients. If a BOOM is used in the immersive setting as viewing device, it is an input device for navigation parameters too (by moving the viewing head and by using the handles). In other system configurations (such as PC clients or non-immersive IRIX clients) the navigation is done with the 3D mouse (or spaceball). The keyboard can be used on all clients for navigation as an alternative to the 3D input devices.

2.4.4 Discussion

Although ICOME presents the user a consistent an intuitive user interface for both immersive and non-immersive clients, a number of problems still arise when working with the application.

The interaction in ICOME mimics real-world interaction, but this interaction misses a number of sensory clue. For example, when putting a vase on top of a table, the vase can pop through this table. In order to overcome this problem, a snapping algorithm that can snap the vase onto the table was implemented, but the problem that the user does not feel this bump and can pass through the table with his hand still remains.

ICOME has this problem in common with a number of virtual environments, such as CHIMP, where the lack of force feedback also posed a problem for precise interaction (Mine et al., 1997).

The rest of this thesis will therefore discuss multi-modal interfaces and will investigate how the multi-modality can bridge this gap between the information that is provided in a virtual environment and understanding that the user has of this virtual environment.

2.5 Multi-modal Interfaces

A multi-modal interface is an interface that makes use of multiple senses for input and/or output. A formal definition is:

> "Multi-modal systems have been developed to take advantage of the multi-sensory nature of humans. Utilising more than one sense, or *mode* of communication, these systems make much fuller use of the auditory, and to a lesser extend, the tactile channel, to improve the interactive nature of the system. Thus multi-modal systems increase the bandwidth of human-computer interaction." (Dix et al., 1998)

Since these multi-modal interfaces increase the bandwidth of human-computer interaction, more information can be presented to the user (or this information is more efficiently communicated). A number of researchers have indeed shown that this assertion is true.

In a survey of design issues in spatial input, Hinckley et al. (1994b) concluded that multisensory feedback should be provided as much as possible since computer-generated worlds can easily confuse users. They further indicate that virtual environments should make use of physical constraints and affordances.

Hoffman et al. (1998) conducted experiments in a virtual kitchen. Participants had either to touch a virtual plate or eat a virtual chocolate bar. The superimposing of a virtual plate onto a real plate was compared with a situation where only a virtual plate was present. Likewise, a situation where a real chocolate bar was present, was compared to a situation where the chocolate bar was only virtual. Their test had the following results for both modalities:

- Tactile cues: people rated the virtual environment as more realistic when they could grab the plate or the chocolate bar.
- Taste cues: tasting something (a chocolate bar) in a virtual environments adds to the realism of the virtual environment.

Dinh et al. (1999) let users walk around in a virtual office suite. They used various modalities and gave the users a questionnaire about their feeling of presence in the virtual environment. Each modality had two possible levels: visual cues could either be high resolution or low resolution; the other modalities were either switched on or switched off. These modalities were audio sensory cues (e.g. sound of a fan), tactile cues (e.g. wind from the fan) and olfactory cues (smell of coffee).

We will shortly discuss the results from this experiments for each modality.

- Visual cues: no effect for the difference between high resolution and low resolution were found.
- Tactile cues: users have a better feeling of presence when they have tactile cues and remember the location of certain objects more easily (e.g. the fan was located in the hall).
- Auditory cues: auditory cues give the users a better feeling of presence and helps to remember the spatial layout of the office (e.g. the location of the bathroom).
- Olfactory cues: olfactory cues, such as the smell of coffee, help the users to remember the location of objects in the office.

From these experiments, we can conclude that the use of multi-modal interfaces gives the user a better understanding of a virtual environment and helps to have a better feeling of presence.

2.6 Summary

This chapter discussed manipulation techniques for virtual environments and elaborated on a few virtual environments. From th experiences with our research project ICOME, it was shown that the usage of multi-modal user interfaces are needed to obtain an intuitive and realistic virtual environment. This claim is supported by other research projects.

The next chapters will discuss how tactile information in general, and more specific kinesthetic information, can be used in virtual environments.

Chapter 3

Passive Touch

Contents

3.1 Introduction

The previous chapter showed that multi-modality is essential for a user in order to get a good understanding of a virtual environment. The next chapters will show how active touch can provide this understanding. This chapter will first elaborate on passive touch .

First, different kinds of passive touch will be explained; next, our implementation of a tangible user interface (a kind of passive touch interface) will be expounded.

3.2 Classification

Over the past few years, different names and definitions for passive touch interfaces have been introduced. This section summarises the three most important terms.

3.2.1 Physical props

The idea of using physical objects to provide feedback in virtual environments has been used for a while. Schmandt (1983) found that real objects can be used for feedback in a virtual environment. The computer generated world was projected on a half-silvered mirror. The user manipulated a physical wand underneath the mirror. However, some problems arose with occlusion effects because the virtual image is always visible upon the user's hand.

Another example is 3-Draw (Sachs et al., 1991), a CAD system for 3D shapes. The goal of this work is to design objects in 3D space, instead of using a 2D interface. The user held a palette in the non-dominant hand and a stylus in the dominant hand. The virtual object was "attached" to the palette. This allowed the user to work relative to the object in his own workspace. This kind of two handed input would also prove to be useful in 2D drawing applications, where the user can manipulate tools with the non-dominant hand and can work relative to these tools with the dominant hand. (Bier et al., 1993; Fitzmaurice et al., 1997; Kurtenbach et al., 1997).

Tognazzini (1993) compared user interface design with the art of magic. One of his ideas is the use of real world metaphors in order to make the user more familiar with the interface. Hinckley et al. (1994a) based their research into passive physical props on the above-mentioned ideas of using physical objects and real world metaphors. The physical props are real world objects that are mapped onto virtual objects. Hinckley et al. used physical props for neurosurgical visualisation of a head. The physical props that they used included a rubber sphere that was used to manipulate the virtual head and a rectangular plate for making slices in the head data.

3.2.2 Graspable User Interfaces

Fitzmaurice et al. (1995) expanded the idea of physical props to a new user interface paradigm: “Graspable User Interfaces”, which they define as:

> “The Graspable UIs allow direct control of electronic or virtual objects through physical artifacts which act as handles for control. These physical artifacts are essentially new input devices which can be tightly coupled or “attached” to virtual objects for manipulation or for expressing action.”

Together with this interface paradigm, they introduced the terms “space-multiplexed” and “time-multiplexed” input devices. Traditional input devices, such as the mouse, are time-multiplexed. For instance, in an GUI, the mouse can first be used to handle a scroll bar and can then be used to handle another scroll. If a physical user interface element is available for each scroll bar, these elements are space-multiplexed and the physical and virtual artifacts blended with each other. Fitzmaurice et al. therefore also define the notion of a graspable object:

> “A *graspable* object is an object composed of both a physical handle and a virtual object.”

In their example Graspable User Interface, Fitzmaurice et al. use physical bricks which are linked to graphical handles in a drawing program. These bricks are approximately the size of LEGO bricks and are placed on a large, horizontal computer display surface.

Fitzmaurice and Buxton (1997) evaluated the effectiveness of graspable user interfaces and found that space-multiplexed input outperforms time-multiplexed input for certain situations, even if the user has to switch between devices.

3.2.3 Tangible User Interfaces

Ishii and Ullmer (1997) introduced the notion of tangible bits:

> “Tangible bits allows users to “grasp & manipulate” bits in the centre of users’ attention by coupling the bits with everyday physical objects and architectural surfaces. Tangible Bits also enables

users to be aware of background bits at the periphery of human perception using ambient display media such as light, sound, air-flow, and water movement in an augmented space."

This theory of tangible bits combines graspable user interfaces with ubiquitous computing: digital bits (virtual objects) are combined with physical objects. This can be compared with graspable objects. Ideally, the presence of the computer should be hidden from the users.

This idea has led to the term "tangible user interface" (TUI), which has replaced the term "graspable user interface". The next section will discuss a few tangible user interfaces.

3.3 Examples of TUIs

3.3.1 TUIs for device interfacing

mediaBlocks

The mediaBlocks projects provides a TUI for capturing, transporting and retrieving digital media (Ullmer et al., 1998; Ullmer and Ishii, 1999). Media-Blocks are small wooden blocks that have an digital ID associated with them. The digital media is thus not stored in the mediaBlocks, but are associated with the mediaBlocks via the digital ID. The media data is stored on different computers, which are connected via a network. The mediaBlocks can thus be considered as a kind of physically embodied "URL".

The mediaBlocks can interface input and output devices that are interfaced with a mediaBlock slot, in which a mediaBlock can be inserted. Different devices that were interfaced are a whiteboard, a printer, a video camera and a video display kiosk. For example, a document is not printed by choosing the print command in an application — it is possible that the user use the whiteboard as input device and does not work with a "classical" computer application — but by putting the mediaBlock associated with the document in the printer's mediaBlock slot.

A last device is a media sequencer device (see figure 3.1), which can build presentations. By associating a part of a presentation, such as a slide or a video fragment, with a mediaBlock, a presentation can be build by putting different mediaBlocks in the right sequence on the media sequencer. This presentation can then be associated with a mediaBlock and played back on the video display kiosk.

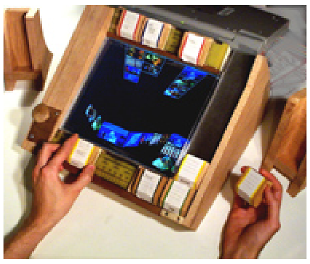


Figure 3.1: mediaBlocks sequencer

IconStickers

A comparable project is IconStickers (Siio and Mima, 1999). An IconSticker is a sticker onto which an icon image, a name and a bar code are printed. If the user drags the icon of a document off the computer's desktop, an IconSticker is printed, which is associated with the icon. These icon stockers can be arranged in physical space. The document is recalled by scanning the IconSticker with a bar code reader.

3.3.2 Build-It

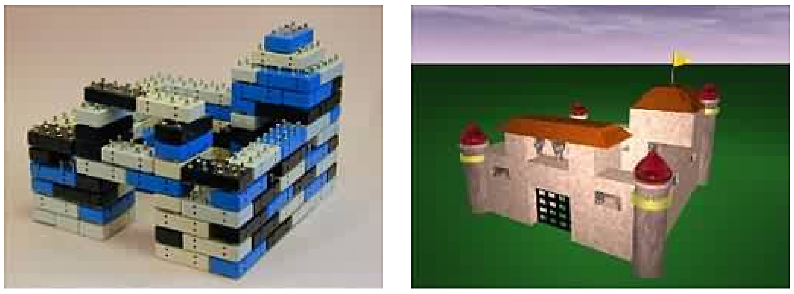
Build-It is an Augmented Reality based planning tool, which presents the user with two views: a table view showing the virtual environment in plan and a wall projection, showing the virtual environment in perspective (Voorhorst and Krueger, 1999).

Objects in the virtual environment are controlled by moving bricks on the table (Fjeld et al., 1999, 2000). The positions and orientations of the bricks on the table are monitored by a camera, mounted above the table. The bricks can be used to select and move objects and to manipulate the virtual camera. The action is stopped by covering the brick. The virtual camera can zoom in and out by moving a second brick relative to the camera brick.

3.3.3 TUIs for 3D Modelling

Anderson et al. (2000) found two interaction metaphors to create 3D geometries: physical blocks and clay models.

The physical blocks can be compared with 2x4 LEGO bricks, but each block contains a dedicated processor. Each block also has eight DC power plugs on the top and eight jacks on the bottom. A geometry can be build be stacking the blocks in a form that resembles the virtual object that is being built. The blocks communicate with their neighbours in such a manner that information of which blocks are connected to each other and with what pins propagates through the model. This information is passed to a computer which recreates geometry and interprets this geometry (see figure 3.2).



(a) Model comprising 118 blocks (b) Interpreted rendering of the model

Figure 3.2: Example model

In another part of this research, 3D geometry is created from clay models. In order to achieve this, the clay models are put on a rotary table and are photographed from different angles. The volume is recreated from these photos.

3.3.4 HandSCAPE

HandSCAPE is a digitally enhanced tape measure (Lee et al., 2000). This tape measure contains electronics to measure the length of the measuring tape and the orientation of the tape measure. This information is sent to a portable computer via radio frequency communication.

Applications of HandSCAPE include optimal packing in storage, modelling architectural interior surfaces and collecting measurements of on-site excavations. In followup research, the software on the portable was adapted for archeological excavations. This research is called GeoSCAPE (Lee et al., 2001).

3.4 Controlling Cut-out Animations

In this section, we will discuss our research into a tangible user interface for cut-out animations.

3.4.1 Cut-out animations

Cut-out animations have been used extensively for the creation of cartoons: a famous example of the use of cut-out animation is Monty Python's Flying Circus. In this technique, animators use photos or drawings, which are cut out in order to form the parts of an animation. Each frame is defined by carefully placing these cut-outs and copying the result onto film. Considering the fact that an animation has a speed of about 25 frames per second in order to obtain fluent motions, it is not surprising that more and more animation companies resort to computer aided cut-out animation packages. These programs allow them to reuse different parts of an animation and to specify movements just by defining various keyframes. Also editing and retouching details of movements are possible, while in the manual process, quite some work has to be redone. The objects to be animated are artworks, either created on the computer — using a paint program or even a 3D rendering program — or hand-drawn images or photos are scanned. Figure 3.3 shows a shot of a boat floating on a rough sea, with different frames of the animation superimposed.

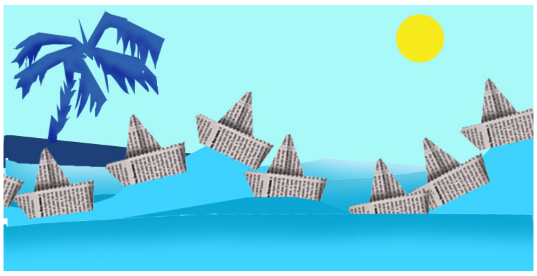


Figure 3.3: Cut-out animation

One of the most important interactions in a cut-out animation application is positioning different parts of the animation and varying this position over time. It is crucial that this can be done in an effective and intuitive way. Many animators still hesitate to use a computer for creating animations, because most animation packages do not offer them a comfortable interaction style. Therefore, we were considering a new interaction paradigm for adjusting positions and orientations in the creation of a cut-out animations by manipulating physical objects. When creating a cut-out animation in the traditional way, animators move objects using subtle hand movements. The animator positions the objects, takes a picture, moves the object a little bit, and takes a new picture. All this in a quick repetitive way, playing the animation in slow motion. The advantage of this way of working is that the animator gets an insight and feeling of the animation in a very natural way. Starting to create this kind of animations via a computer, can be a sudden change: the animator has to revert to interaction devices that are not so intuitive to use. The benefit of editing freedom that the computer brings is partly destroyed by the difficulties encountered in the communication between the animator and the program. Therefore we created an interaction style for creating animations that maps the displacement of physical objects onto the computer animation.

3.4.2 TUI for cut-out animations

Since animators want to interact with the cut-out animation directly using both hand, a tangible user interface is the most intuitive interface for a computer cut-out animation application.

Our cut-out animation TUI, each animation cell that has to be manipulated has a dedicated physical object. These objects are tracked using a camera mounted above the workspace (as depicted in figure 3.4. When manipulating these physical objects, the associated animation cells move on screen.

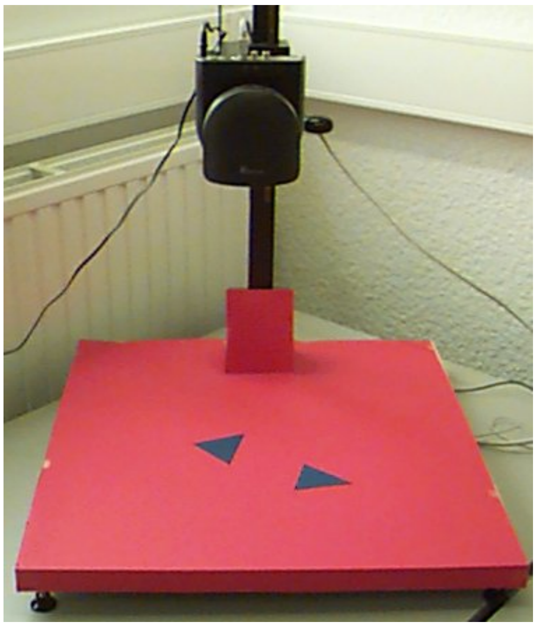


Figure 3.4: TUI setup

3.4.3 Recognising physical objects

In our approach, the tracking of physical objects or the hands that manipulate the objects is crucial. We therefore opted for the use of the camera recognition algorithm, examined by Claes et al. (2000). This algorithm is able to detect shapes in a black & white image that was captured by a camera. We use a three-stage extension of this algorithm:

1. Retrieving data from the image.
2. Data analysis.
3. Assembling shapes.

Before detecting the shapes, using the camera recognition algorithm, one first has to determine which shapes are most suitable. A first requirement for a shape is the unambiguity of its orientation. Shapes, such as ellipses and squares do not have an unambiguous orientation and are therefore not suitable. Furthermore, the shape must be as simple as possible in order for the algorithm to be as fast as possible. We chose isosceles triangles[1] as shapes, because they can be identified from three lines and are defined using two parameters: the height of the triangle and the angle of the base corners. We originally opted for cardboard triangles, but as cardboard bends and tears easily, we employed plastic triangles instead.

3.4.4 Stage one: retrieving data from the image

At a first stage, the sides of the triangle have to be detected. This means that noise, coming from the user's hands, shadows of nearby objects and other artifacts have to be filtered out. Since most of the time, the animator will be manipulating the shapes, differentiating between the hands and the shapes in an early stage of the algorithm will speed up the process.

When looking at human skin, one notices that skin colour contains a significantly smaller amount of blue than red. If some shade of blue were to be chosen as the colour of the triangles, the difference between the user's hand and the triangles would be maximised, making the detection as simple as possible. This has been widely used in television (e.g. a weatherman who is standing in front of a blue screen). Choosing blue as the colour for the triangles enables the easy detection of the difference between the user's hand and the triangles. The difference between the triangle and the table should also

[1]triangles with two equal sides and one different side

be as large as possible. A natural choice for the table's colour is therefore red. We can conclude that blue information in the image belongs to the shapes, while non-blue (red-like) information does not belong to the shapes and is therefore redundant. Recognising edges in the image now consists of detecting the crossings between the red and blue colour components. This results in a new image, where edges are coloured black and surfaces aren't coloured at all. We call this method the “Red-Blue Crossing Method”.

The camera used in the experiments is an inexpensive auto-zoom camera, which is, like most cameras, very sensible for lighting conditions. When a bright light shines on pale skin, the camera detects a near-white colour, causing all other colours to darken considerably. This leads to all sorts of problems that should be avoided in order to get an efficient algorithm. However, when we choose to ignore the contrast in the image, things get better. The crossing between red and blue information gets more distinct and is therefore better recognizably for our algorithm. Since this is an option on the camera, no additional calculations have to be made. Figure 3.5 depicts the difference between the image with and without contrast. We also noticed that the red intensity of the background is much higher than the red intensity of the hand. So when a finger is placed on a triangle, we can rule out the crossing from triangle (blue) to hand (red) and visa versa, because we can just ignore red intensities that are smaller than a certain threshold value.

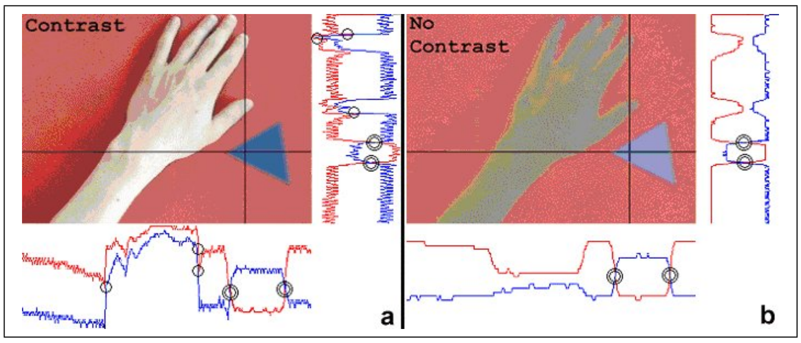


Figure 3.5: Comparison of contrast and no-contrast

Figures 3.5(a) and 3.5(b) depict the Red-Blue Crossing algorithm. Both a horizontal and a vertical line are drawn. The red and blue intensities along these lines are depicted aside and under the pictures. The edges of the triangle in figure 3.5(b) can be easily detected, because of transitions in the intensity lines. When the image contains more contrast, these transitions occur more frequent. The algorithm scans the picture line by line and colours every pixel that contains a transition for red to blue or blue to red is marked with a black colour (the circles in the intensity diagrams). Other pixels are coloured white.

3.4.5 Stage two: data analysis

The result of the first stage is a series of marked pixels. Next, these pixels have to be combined to form lines. An algorithm that is very suitable is the Hough transform (Kälviänen, 1994; Kälviänen et al., 1995; Watt and Policarpo, 1997). Lines can be described by using the normalised equation:

$$ax + by + cz = 0 \tag{3.1}$$

This equation has as drawback that small changes in a parameter lead to bigger changes in some regions of space than in other regions. Therefore we use polar coordinates for the line equation, which leads to more evenly distributed line equations:

$$\rho = x\cos\theta + y\sin\theta \tag{3.2}$$

The Hough transform is depicted in figure 3.6: figure 3.6(a) shows the triangle that has to be recognised. Each line in the original image will be mapped onto a point (ρ, θ) in Hough space. The Hough transform makes use of the dual principle: points in the input image correspond to curves in the Hough space. First, the marked pixels are depicted in Hough space (figure 3.6(b)), with ρ on the horizontal axis and θ on the vertical axis. Every point in the XY space of the input image corresponds to a curve in Hough space: the (ρ, θ) parameters of all lines through the point. Next, dark spots in Hough space are detected; these dark spots correspond with the lines in the original image (figure 3.6(c)). Figure 3.6(b) contains three dark spots. Such a dark spot indicates that their ρ and θ values correspond to many points in the input image. These ρ and θ values define the lines we are looking for.

Although we rule out most noise in the picture with our Red-Blue Crossing Method, noise can still be introduced in the image if the user wears a dark watch or a shirt with blue sleeves. The Hough transform has as a big advantage of being very tolerant to this noise.

3.4.6 Stage three: assembling triangles

After having detected all lines in the image, these lines should be joined to form triangles. Combining three lines, can have three possible results:

1. the lines do not form a triangle (two or more lines are parallel or nearly parallel)

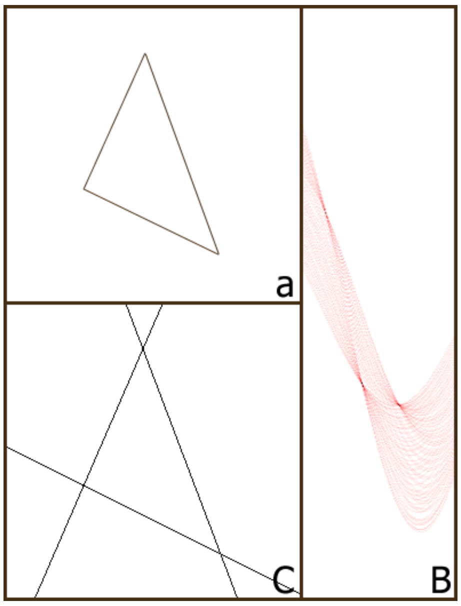


Figure 3.6: Hough transform: A) input image, B) parametric space, C) resulting lines

2. the lines form an arbitrary (not isosceles) triangle
3. the lines form an isosceles triangle

However, not all isosceles triangles that are found, actually exist in physical space. An unlucky combination of three lines of different triangles can form a new isosceles triangle. We call this class of isosceles triangles “ghost triangles”. Figure 3.7 shows all the lines that correspond to two input triangles the user interacts with (the filled triangles) and indicates that this can lead to the detection of a number of isosceles triangles. Three examples of ghost triangles are marked by dashed lines (many more can be found in this figure). This situation happens frequently during an animation session. The section session explains how real triangles can be differentiated from ghost triangles.

3.4.7 Interaction

In the previous sections, we described how the position and orientation of the physical triangles are determined. The next step in the process has to map these parameters onto the position and orientation of animation cells in the computer animation.

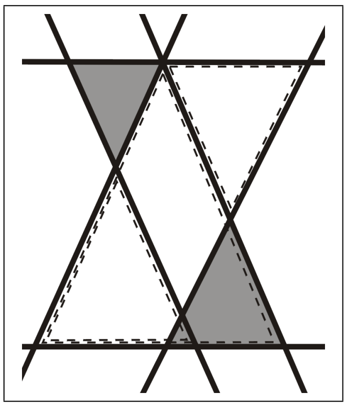


Figure 3.7: Recognizee triangles (real = gray)

A cut-out animation is divided into planes, called layers, in which the objects reside. Every object in a cut-out animation, therefore, has a position that is determined by the layer in which it is situated and its coordinates (x and y) within the layer. Since these objects are 2D objects, they can only be rotated around one axis. The x and y coordinates and the orientation of an object can change during the animation, but typically the object stays in the same layer. Our novel input method provides two coordinates and one rotation angle for each object, hence determining the exact parameters that are needed for the cut-out animation.

During the initialisation phase, the physical triangles are associated with animation cells. From then on, the plastic shapes can be recognised from the grabbed images. After an image is analysed, the position of each triangle is transformed onto screen coordinates and the virtual objects are moved and rotated accordingly. This known-object list is saved between different sessions of the application, so that the user does not always have to initialise the system. It is also possible to add and delete triangles form the known-object list during a session, thus allowing users to determine which triangles they want to use without having to set up the whole system again.

Another advantage of this method is that real triangles can be differentiated from ghost triangles: when analysing the image, the Hough transformation maps the lines in the triangles onto parametric line equations. Three line equations of different triangles can therefore combine to form ghost triangles. One can divide these ghost triangles in two categories: non-isosceles triangles and isosceles triangles. The first category is easily detected by our algorithm. The first feature of an isosceles triangle that's checked are the angles of the triangle. If these angles do not match with a shape in the known object list, the triangle is dismissed, otherwise the height of the triangles is checked against the height of the known object. Most ghost shapes will fail these tests

Several interaction methods with the triangle interface are possible. For instance, all virtual objects receive the exact position and rotation of the corresponding triangles. But also relative movements can be applied: when the program initialises, the virtual objects do not receive the position and orientation of the physical triangles, but the differences between their positions and orientations are taken into account. If a triangle is translated over a certain distance or rotated over a certain angle, this change is added to the current position and rotation of the virtual object. Other possibilities include the association of a triangle with a virtual object that is situated in roughly the same place. Furthermore, also physical constraints between the virtual objects can be applied. These constraints can include a minimum and maximum distance between two objects or restricting some connection angles. Although the objects are not physically linked, the relations between their virtual counterparts enables a powerful control of the animation.

When comparing our method against the two “classic” interaction methods: keyboard and mouse, one can find significant differences. A keyboard can be used to enter the position and rotation of the virtual object. The advantage of this approach is that the input can be very precise. However, entering numbers is not intuitive to use. A mouse can be used to drag the virtual objects to different places in the animation. This is very intuitive for the user. The rotation though cannot be simulated in an intuitive manner. In some animation packages, the rotation is defined by rotating an axis according to the Y movement of the cursor: an increase in movement along the Y axis results in a increase in the angle of rotation. Both solutions share the disadvantage that the user must mentally map the movement of the physical mouse onto the rotation of the virtual object. We believe that these disadvantages do not apply to our solution, because in our interaction paradigm, the action of the user’s hands can be directly mapped onto the changes in the virtual objects’ position and orientation.

3.4.8 Assessment of the TUI

In order to assess the possibilities of the algorithm, an informal user test was conducted. The detection program is realised in an existing real-life application. We opted for the animation package CreaToon, that can be extended using plug-ins: a plug-in was created which can manipulate the position and orientation of a number of animation cells. A number of keyframes in the animation are defined by manipulating the triangles at a particular moment in the animation. Our setup consisted of a camera, which is mounted above a red table. The camera is connected to an image capture board, placed in a PC. For our experiments we used a FlashBus MV image capture board, capable of grabbing a 768x576 image, installed in a Pentium III 550MHz computer with 128MB RAM. However, we only grab 640x480 images, because they can be processed in a more efficient manner. We found that this design choice does not restrict the interaction needed to create convincing cut-out animations. The software is written in C++ and uses the FlashBus SDK to control a SONY EVI-D31 camera.

In this test, ten people used the interaction metaphor in order to define animation paths of animation characters. All of the test persons use a computer on a regular basis, but never used our interaction metaphor before. Some of the test persons already had used a computer animation package though. The goal of this experiment was to test if the algorithm executes a real experiment in real-time and to verify whether the interaction metaphor is really as intuitive to users as we believed.

The first goal was assessed by measuring the time it took to complete one cycle of the image recognition paradigm. Such a cycle consists of the capturing of the image, the recognition of the triangles and the changing of the virtual objects' position and orientation. During the test, the length of such a cycle varied between 12 and 21 milliseconds. We can therefore conclude that our tests were performed in real-time. The reactions and comments of the test subjects were used for assessing our second goal. We noticed that half of the test persons spontaneously used both their hands for the input without being told to do so. All test subjects looked at the screen while manipulating the physical triangles. After the test they reported they felt comfortable with this way of working.

3.5 Summary

This chapter summarised the theory behind passive touch. Furthermore, examples of applications of passive touch were given along with our research in passive touch.

The next chapters will discuss active touch. First, the theory behind active touch, along applications of force feedback will be discussed, next we will demonstrate with formal user experiments that our interaction techniques are valid in haptic virtual environments.

Chapter 4

Active Touch

Contents

4.1 Introduction

The rest of this document elaborates on the active touch, since this is an enabling technique for interaction in virtual environments.

In order to introduce active touch, we first have to define the terminology that is commonly used in active touch research (Oakley et al., 2000):

Haptic: relating to the sense of touch.

Proprioceptive: relating to sensory information about the state of the body (including cutaneous, kinesthetic, and vestibular sensations).

Vestibular: pertaining to the perception of head position, acceleration , and deceleration.

Kinesthetic: meaning the feeling of motion. Relating to sensations originating in muscles, tendons and joints

Cutaneous: pertaining to the skin itself or the skin as a sense organ. Includes sensation of pressure, temperature and pain.

Tactile: pertaining to the cutaneous sense but more specifically the sensation of pressure rather than temperature or pain.

Force Feedback: relating to the mechanical production of information sensed by the human kinesthetic system.

The mechanical generation of stimuli of the haptic system is generally called “haptic feedback”. At this moment, no system is able to provide *full* haptic feedback: cutaneous and force feedback. The reason for this is the complexity of the human haptic system. Designers of haptic feedback devices have to take the human perception into account:

> “Clearly, anyone wishing to construct a device to communicate the sensation of remote touch to a user must be fully aware of the dynamic range of the touch receptors, with particular emphasis on their adaptation to certain stimuli. Its is only too easy to disregard the fundamental characteristics of the human body.” (Kalawski, 1993).

However, the complexity of the human system and the lack of knowledge of this system hinders (at this moment) the realisation of a device that provides all aspects of haptic feedback.

> "A discussion of human tactile psychophysics is complicated by a number of factors, such as the temporal and spatial characteristics of the receptors involved, their saturation (or adaptation) that influences perceived sensations, and the lack of understanding of some related neural paths and cortex mapping. No consensus exists concerning such fundamental aspects as spatial tactile resolution, maximal force exertion, or proprioception resolution." (Burdea, 1996)

Nevertheless, a number of these characteristics, such as maximum sustainable force, spatial and temporal resolution of receptors, ..., can be found in literature (Srinivasan and Basdogan, 1997), although a lot of these characteristics, such as spatial resolution for vibratory stimuli are not yet known (Cholewiak et al., 2001) and current knowledge is based on a few studies of which the results have been obtained using dissimilar experimental setups and methods (Burdea, 1996).

Another factor is the muscles groups that are involved when manipulating a device (or performing a task with a device). Arsenault and Ware (2000) found that performance in a tapping task differs when movements of the forearm (from the elbow) or of whole arm (from the shoulder) are required.

Furthermore, the psychophysics of the user must also be taken into account. An example of a psychophysical effect is the grasp-span weight illusion: when objects of identical mass but different volume are lifted, subjects consistently report that the smaller object is heavier. Davis et al. (2001) showed that this illusion has a purely tactile or kinesthetic cause; it is not caused by the visual interpretation of the object.

The next sections discuss various haptic feedback devices, most of which were designed with the above-mentioned psychophysics in mind.

4.2 Haptic Feedback Devices

This section will discuss a number of haptic feedback devices. Please note that due to the sheer number of existing haptic feedback devices, it is impossible to give a complete overview. Instead, this section discusses a number of representative haptic feedback devices.

4.2.1 Tactile devices

Tactile devices can be compared with braille devices: they consist of a number of actuators (pins) that stimulate the skin. However, tactile arrays have numerous applications beyond braille devices. For instance, van Erp and van Veen (2001) placed actuators on drivers' legs in a car simulation. This setup worked like an in-car navigation display: if the driver should turn left, the actuators on the left leg were stimulated and vise versa. The distance to the turn was indicated by the frequency of the stimulus. Tests in the car simulator indicated that people perform better which a tactile display than with a visual display. This is probably due because the visual information channel is overloaded when driving, while the tactile is not.

A problem that arises with tactile devices is the design of the actuators. Since the actuators should be tightly packed in order to get a good resolution, they should be small and should dissipate as little energy as possible. Furthermore, actuators should be able to bear a significant force. A popular material for actuators are shape memory alloy (SMA) wires (Brenner et al., 2001). A shape memory alloy "remembers" a certain shape. It can be bent into another shape, but when heated it reverts to the original shape. An ideal SMA wire for actuators has two stable states. By applying energy to the wire it switches from one state to the other (or back). This reduces the energy that is needed since an actuator can stay in a certain state for a long time without energy having to be applied.

For a number of applications, the stimulus from an actuator is not constant, but is presented as a waveform stimulus. It is important that this waveform has an appropriate frequency, since this frequency determines which receptor in the skin is addressed. Summers et al. (2001) found that a human can determine the direction for a target moving under his finger if the frequency is 320Hz. Is is possible, however, that another frequency is needed for other tasks.

4.2.2 FEELit Mouse

The FEELit mouse is a mouse which contains motors. These motors can resist the users movement and can thus provide force feedback. The FEELit mouse is developed as TouchSense technology from Immersion Corporation and is commercialised by a number of manufacturers, such as Hewlett-Packard and Logitech.

The usefulness of the FEELit mouse has been tested using a steering task and a combined steering and targeting task (Dennerlein et al., 2000). In both tasks, the user had to steer the mouse pointer through a tunnel. If force feedback was enabled, the mouse would snap to the centre line of the tunnel. In the combined task, the user had to stop and click on a target. A significant benefit was found for the steering task. However, no clear effect was found for the targeting task.

4.2.3 SPIDAR

SPIDAR (SPace Interface Device for Artificial Reality) is a string-based haptic display (Sato, 2001). In this setup, the user's fingers are connected to DC motors via strings and pulleys. The position of the fingers is determined by measuring the length of the strings, while force feedback is provided by changing the tension in the strings.

Different types of SPIDARs exist, ranging from a version which supports 1 finger to a human-scale SPIDAR. All versions support 3 degrees of freedom (DOF) per finger (translations). A recent version (SPIDAR-G) has 6DOF per finger (translations and rotations). Figure 4.1 shows SPIDAR-8, which supports the thumb, index, middle and ring finger of both hands. The demonstration program allows the user the manipulate a virtual Rubik's cube.

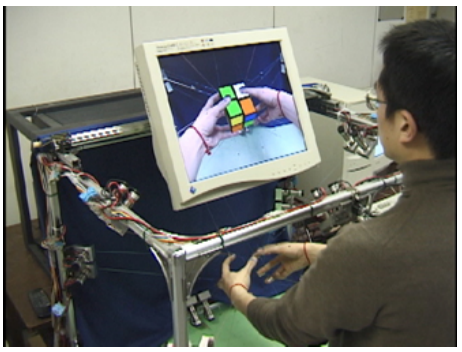


Figure 4.1: SPIDAR-8 (Sato, 2001)

4.2.4 Peltier Haptic Interface

The Peltier Haptic Interface (PHI) is a glove that provides the user with cutaneous feedback: temperature and pressure (Sines and Das, 1999). The PHI is still under development. It uses the Peltier effect to simulate the temperature of virtual objects. This effect is achieved by applying a low DC voltage to a semiconductor-based material. This material act as a heat pump, where the direction of the heat flow depends on the sign of the voltage. The pressure feedback makes either use of piezoelectric and pressure transducers or capillary tubes and micro pumps. The former method uses vibration to simulate pressure, whereas the latter uses fluids.

4.2.5 Rutgers Ankle haptic interface

The Rutgers Ankle haptic interface (see figure 4.2) is a specialised device for ankle rehabilitation (Deutsch et al., 2001; Girone et al., 2001), based on a Steward platform: “a closed loop manipulator whose 6DOF end-effector is connected to a base platform by six actuators using prismatic joints”.

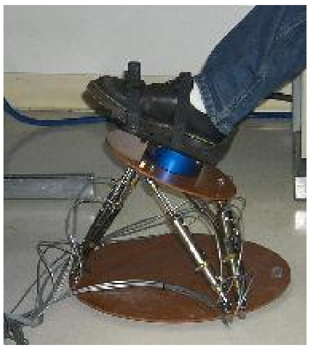


Figure 4.2: Rutgers Ankle haptic interface

In a clinical trial, patients had to fly an airplane through a virtual environment by operating the haptic interface; the movements of the plane corresponded with the movements of the platform. The virtual environment contained a number of squares, where to plane has to fly through. This trials showed that the use of the Rutgers Ankle Haptic Interface is beneficial for patients that need ankle rehabilitation.

4.2.6 Nanomanipulator

The Nanomanipulator (Taylor et al., 1993; Mark et al., 1996) is a teleoperation system that uses a force feedback manipulator arm. This manipulator arm is used to control a scanning tunnelling microscope (STM). A STM is used to image a sample at micrometre scale. The resulting image is a height map of the sample, since the STM measures the distance between its tip and the sample. By applying a current, it is possible to move individual atoms with the STM.

When the user moves the nanomanipulator above the sample, the height of the surface at the tip is sampled. A restoring force is applied to the manipulator arm, moving the arm towards the surface. The user never controls the height of the tip directly. This is controlled by an electronic circuit in order to avoid the user crashing the tip into the sample.

4.2.7 PHANToM device

The PHANToM (Personal HAptic iNTerface Mechanism) device is at this moment the most used force feedback device. It originates from robotics research at the MIT Artificial Intelligence Laboratory (Massie and Salisbury, 1994) and provides point contact forces. Four different versions are available: 3 versions of the original PHANToM ("PHANToM Premium") as depicted in figure 4.3 and 1 version for mainstream applications ("PHANToM Desktop"). These devices all have 6DOF input and 3DOF output. The PHANToM Premium 1.5 also exists in a version that provides 6DOF output, the "PHANToM 6DOF" (Chen, 1999; Cohen and Chen, 1999). The PHANToM Premium versions can either be used with a thimble or with a stylus that has a switch mounted on top. The PHANToM desktop is only available with a stylus. When using the thimble, only 3 translational degrees of freedom are available.

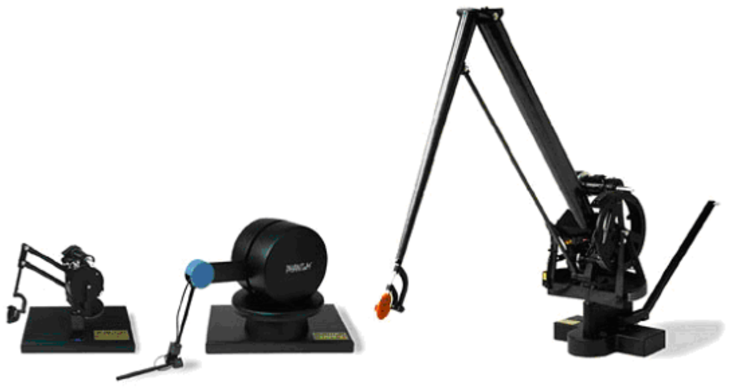


Figure 4.3: PHANToM Premium 1.0, 1.5 and 3.0

Since, we make use of a PHANToM device in our research, the details of programming the PHANToM device will be elaborated on in section 4.3.

4.2.8 Delta Haptic Device

The Delta Haptic Device (Grange et al., 2001) is a force feedback device that is shaped as a delta (see figure 4.4).

The Delta Haptic Device provides 3 translational degrees of freedom; a dedicated plug-in provides an extra 3 rotational degrees of freedom. Like the

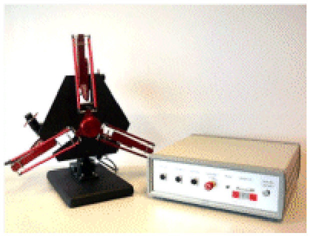


Figure 4.4: Delta Haptic Device

PHANToM device, this device provides point contact forces. The Delta Haptic Device can exert larger forces than the PHANToM device, but its resolution is not as good.

4.3 PHANToM Device

This section elaborates on the PHANToM device and on haptic programming with this device. Table 4.1 summarises the characteristics of various PHANToM devices. These characteristics match the human capabilities which, together with the fact that the interface has traded off complexity with high fidelity, has led to the success of the PHANToM device (Salisbury and Srinivasan, 1997).

	Desktop	Premium 1	Premium 1.5	Premium 3
Resolution	$0.02mm$	$0.03mm$	$0.03mm$	$0.02mm$
Workspace	$16 \times 13 \times 13cm$	$13 \times 18 \times 25cm$	$19.5 \times 27 \times 37.5cm$	$41 \times 59 \times 84cm$
Max. force	$6.4N$	$8.5N$	$8.5N$	$22N$
Cont. force	$1.7N$	$1.4N$	$1.4N$	$3N$
Stiffness	$3.16\frac{N}{mm}$	$3.5\frac{N}{mm}$	$3.5\frac{N}{mm}$	$1\frac{N}{mm}$
Inertia	$< 75g$	$< 75g$	$< 75g$	$< 150g$

Table 4.1: PHANToM device characteristics(Sensable Technologies, 2001b)

Haptic rendering is different from visual rendering (Staples, 1999). A lot of techniques that work in the visual world, do not work in the haptic world. For instance, in order to speed up calculations, some CAD packages tessellate surfaces individually, without connecting the vertices of the meshes when two surfaces are adjacent to each other. This, however, creates gaps in the mesh that are not visible, but are felt with a force feedback device. Another issue is the fact that a high execution rate (>500 Hz) of the haptic loop ("servo loop") is necessary for realistic forces (Salisbury and Srinivasan, 1997). The servo loop is typically executed at a rate of 1 kHz.

A number of API's implement the desired behaviour of the PHANToM device. We will shortly discuss these API's.

4.3.1 GHOST

GHOST (General Haptic Open Software Toolkit) is the PHANToM SDK that is provided by SensAble (Sensable Technologies, 2001a). It is a library of C++ classes that provides a number of objects, such as cones, cubes and polygonal meshes. It also provides a means to make these objects dynamic in order to implement buttons, sliders and dials. Furthermore, GHOST contains classes for creating force fields and effects, such as inertia.

This functionality can be expanded by deriving new classes from the existing GHOST classes. This way, extra objects, effects and force fields (e.g. gravity) can be implemented.

GHOST is only a SDK for haptic programming, but is bundled with a library that provides a visual representation of this haptic world, using OpenGL rendering of the objects.

The GHOST SDK, however, has a number of performance issues. These are explained in detail in section 5.3.2.

4.3.2 VRPN

The Virtual Reality Peripheral Network (VRPN) is a library that provides a device-independent and network-transparent interface to virtual reality peripherals (Taylor et al., 2001). A such, VRPN is not a PHANToM library, but provides an interface to a number of devices.

Since the VRPN interface is a device-independent interface, it does not provide all the functionalities of the PHANToM device. Furthermore, it simulates force feedback by moving up to four planes in the haptic scene graph (Seeger et al., 1997), where other APIs use geometries in the scene graph.

4.3.3 Reachin API

The Reachin API, made by Reachin Technologies, is a API for programming a GHUI, a “graphical haptic user interface”, thus a virtual environment that provides visual and haptic feedback. Programs made with the Reachin API can be extended with Python scripts since this is supported by the API.

The Reachin API also support a co-located stereoscopic display (Evestedt and McLaughin, 2001). This display can be compared with the display of Schmandt (1983), which is explained in section 3.2.1. The Reachin display, however, uses a reflective mirror, not a half-silvered mirror. The user can therefore not see his hand (or the PHANToM device). Instead, an image of a tool is shown on the place where the PHANToM is located. This requires the calibration of the PHANToM device's graphical representation with its physical location and orientation.

4.3.4 e-Touch

e-Touch is an API that is based on the research project FLIGHT (Anderson, 1997; Anderson et al., 1999) and that, like The Reachin API, provides a GHUI. The e-Touch SDK, however, makes use of GHOST for accessing the PHANToM device. All haptic calculations are made within e-Touch (or extensions of e-Touch), but it uses the force field class from GHOST to obtain the PHANToM device's position and orientation and to give the calculated force back to the device.

Because of this method of working, the e-Touch API does not depend on the PHANToM device as force feedback device, but can also make use of other force feedback devices. For instance, ForceDimension provides a driver to use e-Touch for programming the Delta Haptic Device (see section 4.2.8).

The e-Touch SDK is available under a open source-like license.

4.4 Force Feedback in WIMP Interfaces

We will next discuss how force feedback can be used in applications. The rest of this chapter discusses how force feedback can be used in desktop environments, also called WIMP interfaces, which stands for "windows, menus, icons and pointers" (Dix et al., 1998). The following chapters of this thesis elaborates on force feedback in virtual environments.

In section 4.2.2, it was explained that a 2D force feedback device (the FEELit mouse) can be useful in a WIMP interface. The usefulness of a 3D force feedback device, the PHANToM device, is also demonstrated. We will first discuss two implementations of force feedback in WIMP interfaces. The next section discusses our research in this area.

4.4.1 X Windows

The X Window system provides a GUI for UNIX systems. Miller (1998) created a haptic extension of the X Window system by using the GHOST version for SGI IRIX. The PHANToM was constrained to a box of 50mm $\times$ 40mm $\times$ 2mm, which corresponds to the virtual desktop. The slight distance that the

PHANToM can travel in de z direction, allows the desktop to have groves and ridges (Miller and Zeleznik, 1998).

The windows are “haptified” by adding ridges around them. Pushing against such a ridge moves the window in that direction. It is also possible to move a window by pressing against the window and sliding it across the desktop. An window that is obscured by another window can be raised by pulling it up.

Icons are implemented as dimples, thus have a slight ridges around the edges. An icon that is being dragged to a target has a deeper dimple.

Different menu items were separated with ridges. This constraints the pointer to a particular menu item. However, switching from one menu item to another can be very hard, since a ridge has to be overcome.

An informal user test has shown that the addition of force feedback to the X desktop almost doubled the users’ speed during a drag-and-drop task.

4.4.2 MS Windows

Contrary to X Windows, no integration of haptic feedback in the Microsoft Windows environment has been realised. However, research into haptically enhanced widgets in MS Windows has been performed.

Oakley (1999) added four different haptic effects to a button: a haptic texture, friction, recess and a gravity well. The friction allows the user to feel the button because it adds a series of concentric ridges to the button. Friction damps the user’s velocity when the pointer moves over the button. A recess is a hole in the back of the workspace, where the button is located. A gravity well draws the pointer to the middle of the button. A formal user test was conducted in order to asses the affectivity of these enhancements (Oakley et al., 2000). In this test, the different effects and a non-haptic button were compared. Users made more errors in this test with the texture button. The user were also not satisfied with this effect. The gravity well, and to a lesser extent the recess effect, were significantly better than the other buttons. Little difference between the friction button and the non-haptic button was found.

A comparable research involved a haptic slider (McGee, 1999). This slider was enhanced with a gravity well in the up and down arrow buttons and a recess effect in the slider area. A formal user test showed that the haptic effect did not cause the users to be faster. However, the users moved the pointer less on and off the scroll bar area in the haptic condition. The mental workload did also decrease.

4.5 2D Curve Creation with a 2.5D Pen Metaphor

The above-mentioned research into 2D interaction focussed on general principles to incorporate haptics in the interfacing elements (widgets) of WIMP interfaces. However, specialised interface may need specialised interaction metaphors. We therefore investigated how haptic feedback can be used in a curve drawing tool. This tool has both been used for drawing images (Raymaekers et al., 2000) and making quick sketches (Raymaekers et al., 2002). An example of such an image can be seen in image 4.5.

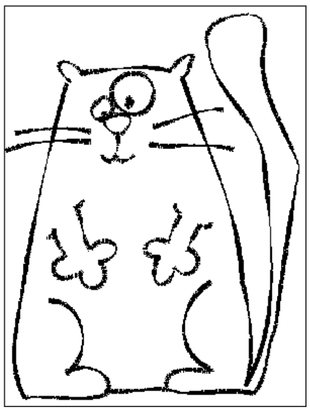


Figure 4.5: Example image

When creating two-dimensional curves, a graphics tablet is used most of the time, because for most people it is a natural and intuitive interaction style and it is therefore preferred over the use of a mouse. An extension of this interaction device is the pressure-sensitive graphics tablet, because it provides a means to vary the width of the curve that is drawn and acts as a physical haptic constraint. However, when editing or deforming a curve, haptic feedback can be used in order to give the user an idea of the implications of her actions. We opted for a PHANToM device with an encoder stylus instead of a two-dimensional force feedback device. The PHANToM device incorporates both the advantage of the pen-like interface and of haptic feedback in horizontal as well as in vertical dimension. So the PHANToM device allows us to combine the best of both solutions. Using force feedback, it is even possible to mimic different types of drawing surfaces, like paper or a blackboard. We should stress that we do not use the term 2.5D as it is mostly used in computer graphics: an animation type where 2D figures are placed in layers with different depth, thus creating a 3D effect with 2D figures. We define our interaction metaphor as 2.5D, because we both use the PHANToM device as a 2D and a

3D device. The makeup of our curves is completely determined by 2D information: they reside in 2D space; the width of the curves, however is determined by the pressure that is built-up against the virtual drawing plane. This can be determined by measuring the depth value of the PHANToM position. Our interaction paradigm can therefore not be cataloged as 2D or 3D.

The majority of recent curve drawing research projects focuses on all three dimensions. Cohen et al. created a tool for drawing 3D curves. Igarashi et al. created a drawing tool that creates 3D objects from 2D curves. These objects can further be used in other applications (Conway et al., 2000). Recently, Personal Digital Assistants (PDA's) are becoming more and more popular. These devices use a pen as input device, which has led to increased research into pen gestures (Long et al., 2000; Mankoff et al., 2000).

Cartoonists and animators however often want to create 2D cartoon figures. Two-dimensional curve drawing techniques are more suitable for this work. Some curve drawing and curve fitting techniques can be found in (Schneider, 1990; Baudel, 1994).

We were, however, not entirely satisfied with current curve drawing techniques. We wanted to create a tool which allows our users to define mathematically correct curves in an intuitive manner. It is crucial for our work that these curves are mathematically correct, because it allows our software to interact with the curves in a more efficient manner. Most users are not mathematically skilled, so the mathematics should be entirely hidden from the user. Therefore, an animator helped us during the development of our algorithm and gave us feedback about the intuitiveness of the input mechanism.

4.5.1 System overview

The system allows for the creation of curves, which are represented by cubic Bézier splines (Foley et al., 1990). The splines are created and modified in the graphics loop, while the forces are generated in the GHOST haptic loop, as depicted in figure 4.6. The forces are fed back to the PHANToM device, using a class that is derived form a gstForceField (Sensable Technologies, 2001a). A gstConstraintEffect is used to limit the PHANToM movement to the XY plane.

4.5.2 Creating curves

A new curve is generated in real-time whenever the user presses the PHANToM button. As long as the button is depressed, new points are added to the curve. We must stress that our curves consist of a series of segments, where each segment is represented by a cubic Bézier spline (see figure 4.7).

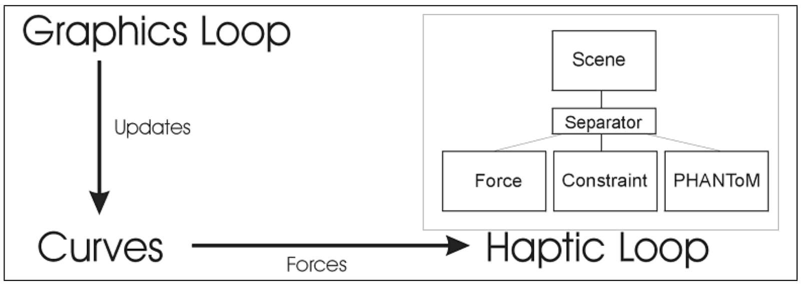
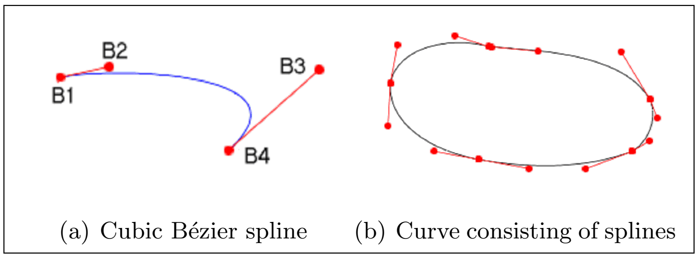


Figure 4.6: System overview



(a) Cubic Bézier spline (b) Curve consisting of splines

Figure 4.7: Curves

Each haptic loop, where the position of the PHANToM device corresponds to a new (x, y) position, a new data point is created. The current spline is refitted to contain the new data by minimising the least square error (Press et al., 1995) between the spline and the data. If this value stays under a certain threshold, the spline segment is adjusted; otherwise a new spline is created. The smoothness of the curve can be controlled by adjusting the number of data points that are shared between two successive spline segments.

Adding points

In order to determine the spline, we first notice that a cubic Bézier spline is represented by the following equation:

$$f(t) = (1-t)^3 B_1 + 3t(1-t)^2 B_2 + 3t^2(1-t) B_3 + t^3 B_4 \tag{4.1}$$

We must find the control points (B_1, B_2, B_3, B_4) so that the difference between the spline and the corresponding data points $(P_1, P_2, \ldots, P_n)$ is as small as possible. Considering the above-mentioned smoothness, represented by parameter s, the first and last control point can be chosen as:

$$\begin{aligned} B_1 &= P_s \\ B_4 &= P_{n-s} \end{aligned} \tag{4.2}$$

The two other control points can be constructed using the least square minimisation method. In order to use formula 4.1, the t-values must be calculated for each data point. First the distance between two successive data points is calculated, so that we can calculate the total length of the segment.

$$d_{i,j} = |P_i - P_j| \tag{4.3}$$

$$L = \sum_{i=s}^{n-s-1} d_{i,(i+1)} \tag{4.4}$$

We can now calculate the t-values and use them in the least square minimisation:

$$\begin{aligned} t_s &= 0 \\ t_i &= \frac{\sum_{j=s}^{i-1} d_{j,(j+1)}}{L} \qquad \forall i > s \\ t_i &= \frac{\sum_{j=s}^{i+1} d_{j,(j-1)}}{L} \qquad \forall i > s \end{aligned} \tag{4.5}$$

$$\begin{aligned} S &= \sum_{i=1}^{n} (P_i - f(t))^2 \\ \frac{\partial S}{\partial B_2} &= 0 \quad \frac{\partial S}{\partial B_3} = 0 \end{aligned} \tag{4.6}$$

Substituting equations 4.3, 4.4 and 4.5 in equation 4.6, yields the following equation:

$$\begin{cases} B_1 \sum t_i(1-t_i)^5 + 3B_2 \sum t_i^2(1-t_i)^4 + 3B_3 \sum t_i^3(1-t_i)^3 + \\ \qquad b_4 \sum t_i^4(1-t_i)^2 - \sum P_i t_i (1-t_i)^2 = 0 \\ \\ B_1 \sum t_i^2(1-t_i)^4 + 3B_2 \sum t_i^3(1-t_i)^3 + 3B_3 \sum t_i^4(1-t_i)^2 + \\ \qquad b_4 \sum t_i^5(1-t_i) - \sum P_i t_i^2 (1-t_i) = 0 \end{cases} \tag{4.7}$$

The values of B_2 and B_3 can be determined by solving equation 4.7.

Controlling the Width

A planar constraint effect is used in order to restrict the movement of the PHANToM device in the z direction. However the stylus can be moved slightly in the z direction by exerting a force in this direction. This information is used to determine the width of the curve: the harder the user pushes, the wider the line is drawn. In contrast to the position of the data points, which are determined using Bézier interpolation, the line width at the points is calculated using a linear interpolation. The line width of the curve is thus represented as set of interconnected line segments.

$$W = (1 - t_w)W_1 + t_w W_2 \tag{4.8}$$

W_1 and W_2 represent the width at the start and end of the segment. The t_w-value, which is used to determine the width at the point, can be determined using the t-value which is used to calculate the position of the point, as described in the previous section. But these values do not necessarily have to be equal, which means that a Bézier segment can contain more than one width segment. The calculation of the line segments is based on a least square error. When the calculated error is larger than a certain value, a new segment is created. The W_1 and W_2 values of the new segment are equal to the W_2 value of the last segment, which provides a smooth transition of the line width; otherwise the W_2 value of the current line segment is set equal to the width of the last data point.

4.5.3 Modifying curves

Sometimes, the user is not entirely satisfied with the image that is made. Instead of having to redraw the image, local modifications can be made. Since we make use of cubic Bézier splines, the problem of modifying an image can be reduced to moving the control points of the segments. However, most people are not familiar with cubic Bézier splines and find it hard to get the desired result. Therefore, our tool lets the user pull and push against the curve. The control points are transformed transparently in such a manner that the modification of the image mimics the user's movement, thus allowing to intuitively redraw part of the sketch. We do this by sketching the modification in the neighbourhood of the curve: the curve is pulled towards the pointer. This process consists of three steps and is repeatedly executed during the modification stage.

Selecting a Point on the Curve

To select the point nearest to the pen position we first need to determine the nearest Bézier segment. Since our tool makes use of OpenGL for rendering, the OpenGL picking functionality can be used (Woo et al., 1999). Next, the selected point on the Bézier spline is determined by calculating the above-mentioned t-value. This t-value is determined so that the distance between the PHANToM position and the corresponding point is as small as possible.

The distance between the PHANToM position (x, y, z) and the spline is given by:

$$g(t) = (x - f_x(t))^2 + (y - f_y(t))^2 + (z - f_z(t))^2 \tag{4.9}$$

The desired t-value is found by solving the following equation for t:

$$\frac{\partial g(t)}{\partial t} = 0 \tag{4.10}$$

Equation 4.10 is a 5th degree polynomial in t. Solving such an equation results in up to five different solutions for t. Only one of these solutions is a real number in the interval $[0, 1]$. We use Laguerre's method in order to solve equation 4.10 because the first solution, which satisfies these constraints, that is found by the algorithm is the value we are looking for. If no such value is found, the selection mechanism fails, which means that and no point is selected on the curve.

Modifying the Curve

Next, a vector $\overrightarrow{V}$ is defined:

$$\overrightarrow{V} = (x, y, z) - (f_x(t), f_y(t), f_z(t)) \tag{4.11}$$

To adjust the Bézier segment, the vector $\overrightarrow{V}$ needs to be distributed over its control points. The influence of $\overrightarrow{V}$ on each of the control points is determined by the basis functions for a cubic Bézier spline. So the new control points $B_1^{'}$, $B_2^{'}$, $B_3^{'}$ and $B_4^{'}$ are given by:

$$\begin{aligned} B_1^{'} &= B_1 + (1-t)^3\overrightarrow{V} \\ B_2^{'} &= B_2 + 3t(1-t)^2\overrightarrow{V} \\ B_3^{'} &= B_3 + 3t^2(1-t)\overrightarrow{V} \\ B_4^{'} &= B_4 + t\overrightarrow{V} \end{aligned} \tag{4.12}$$

In order to keep the smoothness at the transition from one segment to another, the control points from the previous and next segment will also be moved. We can distinguish three different cases for the values of t:

1. $t \approx 0$: Both the control points B_1 and B_2 of the selected segment as well as B_3 and B_4 of the previous segment will be moved over the distance $\vec{V}$.

2. $t \approx 1$: Both the control points B_3 and B_4 of the selected segment as well as B_1 and B_2 of the next segment will be moved over the distance $\vec{V}$.

3. $t \in]0, 1[$: B_2 and B_3 of the selected segment are moved over $\frac{\vec{V}}{3t(1-t)}$. The smoothness is preserved by rotating B_3 of the previous segment around B_1 of the selected segment and by rotating B_2 of the next segment around B_4 of the selected segment.

Haptic Feedback

When modifying an image, a force is fed back to the user which resists the movement. This force counterbalances the movement of the curve: the pointer is drawn to the curve with a force ($\vec{F}$) proportional to the force with which the curve is pulled to the pointer:

$$\vec{F} = -l(\vec{V}). \tag{4.13}$$

The function l scales the magnitude of $\vec{V}$ in such a manner that equation 4.14 is true.

$$\begin{aligned} &0.2N \leq \|\vec{F}\| \leq 1N \\ &\|\vec{V}_1\| < \|\vec{V}_2\| \Rightarrow \|l(\vec{V}_1)\| < \|l(\vec{V}_2)\| \end{aligned} \tag{4.14}$$

This force reduces errors by making large movements harder to perform. On the other hand the forces that are generated are never so big that they hinder (or fatigue) the user.

4.5.4 Transformations

Most vector drawing editors provide a selection tool and tools to transform selections. Likewise, our tool provides a lasso selection tool. Unlike bitmap editors, the tool doesn't select the pixels within the selection area, but the control points within the selection area. This allows for a more efficient manipulation of the part of the image in which the user is interested.

A number of tools are implemented which transform the segments of which one or more control points are selected. The selected area can either be translated, rotated or scaled down. Because these tools work on the cubic Bézier splines and not on individual pixels, the connectivity between the curves is kept.

Again, force feedback can help when performing these editing tasks: by defining different effects to counteract the modifications, the user is guided during the manipulation task. The force that counteracts the transformations is dependent of the curves that partially belong to the selection. More exactly it is dependent on the first control points (B_1 or B_4) along the curves who are not part of the selection. This way the force increases with the amount of curves that connect the selected with the non-selected part of the image. This makes it harder to move a selection that has a tight connection with its surroundings.

Translation

The force is a scaled sum of the translations of the mentioned control points:

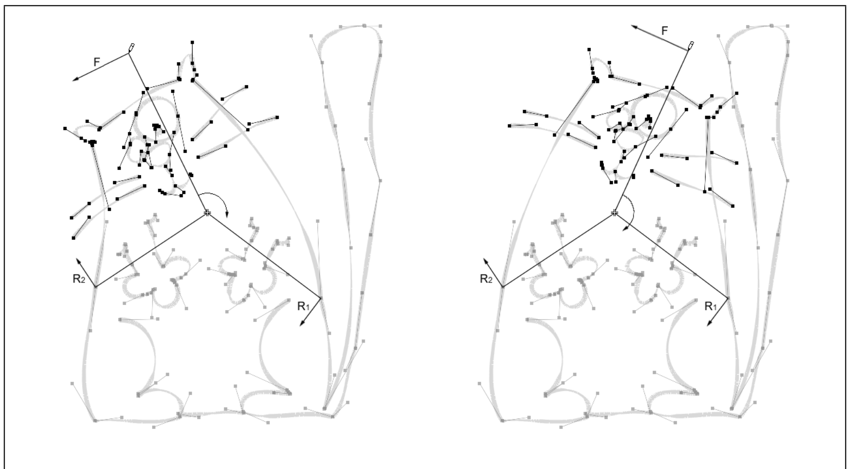
$$\begin{aligned} \overrightarrow{V}_1 &= (x, y, z) - B_1 \\ \overrightarrow{V}_4 &= (x, y, z) - B_4 \\ \overrightarrow{F} &= -l(\overrightarrow{V}_1 + \overrightarrow{V}_4) \end{aligned} \tag{4.15}$$

Rotation

The size of the force is a scaled sum of the alleged translation of the mentioned control points. The direction is perpendicular to the vector from the centre of the rotation to the position of the pen. The calculation of the counteracting force is depending on the type of transformation, as depicted in figures 4.8(a) and 4.8(b), which depict the rotation force at two moments during the interaction (rotating the head of a cat).

Scale

The force is a scaled sum of the alleged scale of the mentioned control points. These calculations are analogous to the calculations in the translation case.



(a) Rotation force at a point in time (b) Rotation force, a few seconds later

Figure 4.8: Example of the forces when rotating(the pointer is indicated by the pencil in the upper left corner)

4.5.5 Additional Functionality

Next to creating a curve and modifying its position, we can also define a wide range of additional tools, which can also be implemented without exposing the mathematical structures to the user. In this paragraph we will shortly describe some of these tools.

Eraser

Erasing a part of an image can be done by selecting the nearest point to the cursor with each move of the eraser. Subdividing the Bézier segment and width segment at this point allows us to delete the part of the segment traversed by the eraser without any changes to the rest of the image.

The acting force works the same as with the creation of a curve, only with a different scale according to the difference in material (pencil versus eraser). This effect is achieved by altering the resistance of the haptic plane. A pencil moves more easily over a piece of paper than an eraser. An increase in friction mimics this effect and decreases the number of errors that are made because the user has more difficulties to move the pointer if the friction is larger.

Connect Curves

An additional property of the drawing tools is the possibility to automatically connect the start or end points of the newly drawn curve to neighbouring curves. With this addition it is also possible to create closed curves.

The end point of the curve is snapped towards the begin point if the pointer (thus the end point) comes in the vicinity of the begin point. Likewise, the user feels an attracting force towards the begin point:

$$\begin{aligned} \overrightarrow{V} &= P_0 - (x, y, z) \\ \|\overrightarrow{V}\| < M &\Rightarrow \overrightarrow{F} = l(\overrightarrow{V}) \end{aligned} \tag{4.16}$$

The begin point is indicated by P_0, the pointer position by (x, y, z) and l is the function that was discussed in the paragraph about editing. The parameter M defines a circle in which the effect is felt. We can hereby control if the user should feel the force before the curve is snapped to the begin point.

This force also allows us to turn off the snapping of the image. It is possible that the user wants the image near the begin point, but does not want the image to snap. If only the force is active, but not the image snapping, the user can avoid this snapping effect by simply resisting the force.

Change Curve Width

Previously, a tool for correcting the position of the curve was described. It is also possible to modify the width of the curve. This is done again by selecting the nearest point to the cursor on the curve. But instead of changing a Bézier segment, the width segment will be modified by increasing or decreasing the width at its start and/or end point. It is also possible to subdivide the line segment when more level of detail is needed.

The size of the width modification is dependent on the pressure in the z direction exercised by the user. The maximum width is limited because the pressure of the user against the plane is counteracted by a spring-damper system.

4.6 Summary

This chapter introduced active touch and showed how force feedback can be used in 2D interaction. Since the goal of force feedback devices is to provide a natural means to work with computer-generated worlds, the rest of this document focusses on virtual environments.

Chapter 5

Haptic Feedback in Virtual Environments

Contents

5.1 Introduction

The previous chapter discussed a number of application areas of force feedback. This chapter elaborates on the usage and importance of force feedback in virtual environments. Some examples of accomplishments in virtual environments will be given. Special attention will be given to the haptic rendering techniques that make objects touchable in a virtual environment. We will further show how force feedback has been integrated in the framework of section 2.4 and explain how the user is supported in this new framework.

5.2 Application Areas

Due to the novel nature of haptic feedback, few research into the full integration of the force feedback into virtual environments has been performed. Research areas are specific interaction techniques and special-purpose virtual environments. We will discuss these application areas before we will expound on haptic rendering techniques.

5.2.1 Medical applications

A lot of applications of force feedback are medical applications. One example is rehabilitation of patients. Section 4.2.5 about the Rutgers Ankle haptic interface discussed how patients with ankle injuries can benefit from force feedback for rehabilitation. Force feedback is also used for rehabilitation of stroke patients. One of the biggest problems in this kind of therapy is the patient's motivation to keep exercising. Loureiro et al. (2001) found that the use of a VR system with force feedback increases patients' motivation.

A lot of effort has been spent on training and simulation of surgery. An important aspect of this application is the haptic rendering of tissue, since the rendering of tool-tissue interaction (De and Srinivasan, 1998) and soft tissues (Balaniuk and Costa, 2000) is complex but important to achieve a realistic effect.

Early research into surgical trainers looked at the possibility of using force feedback in surgery training. In two independent investigations, surgery trainers were developed (O'Toole et al., 1997; Gorman et al., 1998). The efficiency with these trainers of both experienced surgeons and medical students was assessed. This research indicated that experienced surgeons performed better, but it also showed that force feedback is beneficial for surgical training.

Furthermore, a number of specialised trainers have been developed. Ranta and Aviles (1999) developed a trainer for dental students, since existing plastic models of teeth and jaws lack the level of detail and material properties needed to simulate real teeth. This problem was overcome using force feedback in a virtual environment.

Another example is the simulation of catheters (Zorcolo et al., 1999). Informal testing by a surgeon showed that this simulation is realistic when working with soft tissues. The rendering of hard tissues can be problematic with present day force feedback devices (see also section 5.3.2).

More difficult procedures are also simulated. A bone marrow harvest simulator (dos Santos Machado et al., 2000) was developed. This skill is important for transplantations that can cure diseases such as leukemia.

Veterinary students also benefit from haptic feedback in virtual environments. Crossan et al. (2000) developed a horse ovary simulation palpation simulator. This simulator allows veterinary students to practice a procedure that is uncomfortable and potentially dangerous for horses.

5.2.2 Data visualisation

A second application is data visualisation[1]. This can be the visualisation of medical data. Visualisation of data obtained by a medical scanner has been investigated (Mor et al., 1996). Another application is the "visible human project". This is a sub-millimetre photographic and radiological description of both a male and female corpse. Reinig (1996) created algorithms to haptically render the visible human.

Non-medical visualisation has also been investigated. McLaughlin and Orenstein (1997), for instance, visualised off-shore seismic data. This allows people to feel data that can not always be seen, since it is visually obscured by

[1]Please note, that although the research does not concentrate on making the data visible, but touchable, this domain of haptic rendering is still called visualisation.

other information in the data set. This application is useful for the petroleum industry.

Force feedback can also be used to aid the visually-impaired. Experiments showed that force feedback allows people to feel graphs (Yu et al., 2000, 2001), while sound and haptics can display mathematical functions to visually impaired people (Van Scoy et al., 2001).

5.2.3 Molecular modelling

Since forces are important in chemical models, force feedback can be used to enhance molecular modelling applications. Wanger (1998) incorporated haptic feedback into a commercially available molecular modelling application. Křenek looked at molecular flexibility (Křenek, 2000) and molecular conformations (Křenek, 2001), which are important in the modelling of biological molecules.

5.2.4 Virtual prototyping

Some research into haptic prototyping has been done. The LEGOLAND project used force feedback to make models with virtual LEGO blocks (Young et al., 1997). Anttilla (1998) used force feedback for prototyping of handheld products, such as cell phones.

5.3 Haptic Rendering Techniques

In order to make virtual objects touchable, haptic rendering techniques have to be applied. Two classes of rendering techniques can be distinguished: volumetric and geometric rendering. The usage of the above-mentioned rendering techniques depends on the nature of the data and the application.

5.3.1 Volumetric rendering

A number of techniques exist that can make volumetric data touchable. Most of these techniques divide the data set in a series of voxels. Next isosurfaces are created by applying algorithms such as the marching cubes algorithms (Avila, 1999a; Bartz and Gürvit, 2000). These isosurfaces are mostly rendered using geometric rendering techniques as discussed in the next section.

The application areas for volumetric rendering range from assembly and path planning (Avila, 1999b) and geometric modelling (Savchenko, 2000) to (medical) simulations (Azouz and Payandeh, 2001). Since the majority of applications require geometric information, most haptic API do not use volumetric rendering algorithms. Virtual environments also make use of geometric representations of object. The rest of this chapter will thus discuss and make use of geometric rendering algorithms.

5.3.2 Geometric rendering

Most haptic implementations are based on the rendering techniques that are developed by Zilles and Salisbury (1995) and Ruspini et al. (1997). These rendering schemes calculate a point on the surface of the object where the user has penetrated the object. This point, called the surface contact point (SCP)[2], is the point on the surface that is closest to the position of the interaction pointer. Next, a virtual spring is attached to the SCP and the object representing the haptic device, which pulls the user towards the surface contact point as can be seen in figure 5.1. In order to avoid oscillations of the SCP a spring-damper system is used. This approach is called the "constraint based method" because it constraints the user's position onto the object's surface (Ruspini, 1999). In reality the user is not restricted to the surface, but can penetrate the object. Fortunately, the human perception is not sensitive enough to notice a slight difference between the real position of a finger and the visual representation (the SCP).

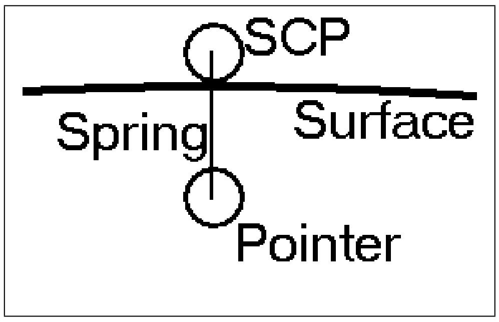


Figure 5.1: Surface Contact Point

A disadvantage of this approach is the fact that solid objects can feel a bit mushy. This can be solved by adjusting the spring parameters. However, since a spring-damper system is a partial differential equation, the simulation becomes unstable if the damping constant is too large for a given step size, which is usual 1ms as explained in section 4.3. In order to render stiff walls a

[2]In earlier publications the SCP has a number of other names such as the "proxy" and the "god object".

higher frequency of the servo loop must be obtained. Experiments have shown that a haptic rate of up to 6kHz must be used in order to get a stable rendering of stiff walls. This requires the use of a real-time operating system (Kabeláč, 2000).

The haptic rendering algorithms for most geometric primitives is known for some years. The GHOST SDK for instance implements cones, cubes, cylinders, spheres and toruses (Sensable Technologies, 2001a). Due to the 1kHz constraint, the algorithms that are performed in the haptic loop have to be very fast. In fact, in order to allow other processes to execute, the algorithms for reading the haptic device's encoders, calculating the forces and applying these forces, have to be executed in less than 0.9ms. However, due to increasing functionality, and thus added calculations, with each version of the GHOST SDK, the number of non-overlapping primitives that can be simultaneously rendered is lowered for newer versions of the SDK. Acosta and Temkin (2001) measured for each version the highest scene complexity for which an application always runs with stability and no errors (the greatest lower bound, GLB) and the lowest scene complexity where instability and errors always occur (the least upper bound, LUB). Their results of this test on a Pentium III 500 MHz computer is summarised in table 5.1.

Version	Object	GLB	LUB
1.2	Box	1100	1188
	Sphere	1100	1200
2.0	Box	800	847
	Sphere	770	847
3.0	Box	600	729
	Sphere	600	729

Table 5.1: GLB and LUB for GHOST versions

The drawback of these primitives is the fact that they can only describe simple objects. The GHOST SDK therefore also implements a polygonal model. This polygonal model, however only allows a limited number of polygons. Acosta and Temkin report a GLB of 120.000 polygons and a LUB of 235.000 polygons when only one object is present in the scene. When working with objects consisting of 1.200 polygons, they found a GLB of 35 and a LUB of 49 objects. This polygon count is insufficient to render a number of complicated objects.

The polygon model which is used in e-Touch (see section 4.3.4), called the "ActivePolgon" polygonal algorithm, makes use of an octtree representation of the object (Anderson and Brown, 2001). The performance of this rendering algorithm is almost independent of the number of polygons in the object. Table 5.2 shows the results of a test by Anderson and Brown, which was performed on a dual Pentium III 800MHz computer.

SDK	Polygon count	Haptic load avg.	haptic load peak
GHOST	5.706	20%	30%
	134,232	85%	100%
	239,694	70%	70%
	1,063,452	unstable haptics	
e-Touch	5.706	20%	20%
	134,232	15%	20%
	239,694	15%	20%
	1,063,452	15%	80–100%

Table 5.2: Comparison between polygon implementations

The algorithms of the GHOST SDK and e-Touch were assessed by looking at the haptic load when making the calculations. The haptic load is defined as the percentage of the processor time that is needed to perform the haptic algorithms. This haptic load includes (but is not limited to) the time needed to read the haptic device's encoders, to evaluate the haptic scene graph and to sent force back to the device. Since the haptic loop must be performed at a rate of 1kHz, the haptic load is a considerable percentage.

Unfortunately, at this moment, no exact measurement of the haptic load is available since the haptic loop is shielded from the user (one can only write extensions of the haptic loop). The PHANToM device drivers, however, include a utility that visually shows the haptic load. Since this is the only way to assess the haptic load, this tool is currently considered a valid (although not precise) means to measure the performance of haptic algorithms.

In computer graphics, a number of object representations are used, depending on the application. In order to use these object definitions, haptic algorithms must be created that make these objects touchable. Next, we will elaborate on our haptic algorithms for subdivision surfaces (as reported on in (Raymaekers et al., 2001a)) and for CSG trees.

5.4 Subdivision Surfaces

A subdivision surface is a surface that is defined as the limit of a series of refinements $M_1, M_2, \ldots$, starting from an original control mesh M_0. Because of this property, they support level-of-detail. Since they can also be used to efficiently represent objects of arbitrary topology, and they can be modified easily to support features such as creases and boundaries, it is no surprise subdivision surfaces are already used in computer graphics and computer animation. The most important examples are Pixar's short animation "Geri's Game" (see figure 5.2) and feature animation "Toy Story 2".

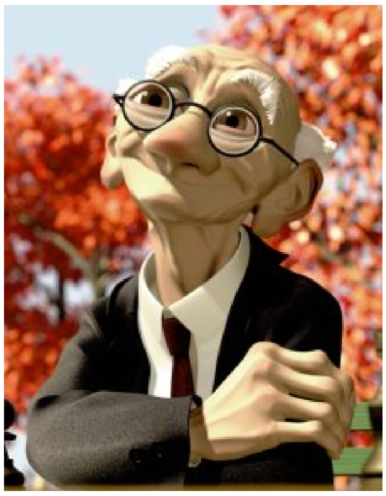


Figure 5.2: Geri's Game ©1997 Pixar Animation Studios

Subdivision surfaces have as advantage that they can describe arbitrary topologies in an efficient manner. Furthermore, they support animation deformations. If for instance, a NURB would be used, the seams of the patchwork can become visible during deformation (DeRose et al., 1999).

The subdivision process consists of two stages. In the first stage, known as the splitting stage, a new vertex is added in the middle of each edge, and both the old and new vertices are connected to form new polygons for each old triangle. In the smoothing stage, all vertices are averaged with their surrounding vertices. Before elaborating further on subdivision surfaces, we will first introduce subdivision curves in order to better explain the subdivision process.

Subdivision curves

A subdivision curve is a curve that is defined as the limit of a series of refinements $C_1, C_2, \ldots$, starting from an original control curve C_0. Again a splitting and averaging step is used to create the refinements. Figure 5.3 shows an example of a subdivision curve. This subdivision process uses the subdivision mask $\frac{1}{4}(1,2,1)$, which indicates that each vertex is replaced by the average of the position of its neighbours and twice its own position.

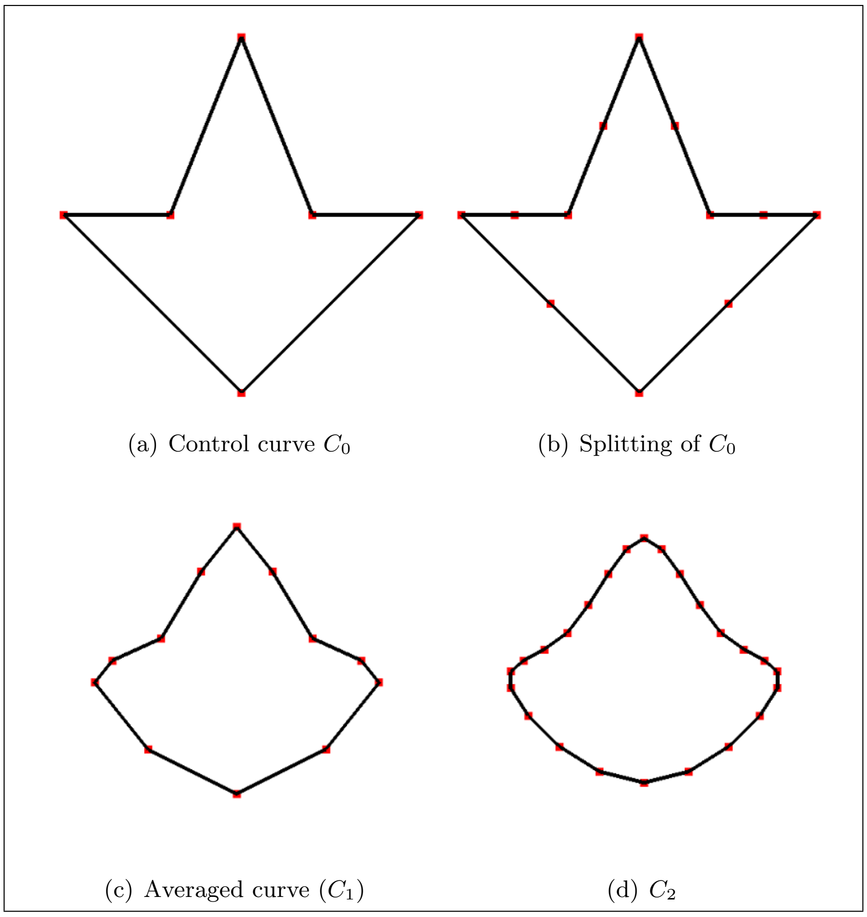


Figure 5.3: Example subdivision curve

Subdivision schemes

A wide variety of subdivision schemes for surfaces exist, with an equally large variety in properties. Two of the most well-known subdivision schemes for surfaces are the Catmull-Clark scheme (Catmull and Clark, 1978; Halstead et al., 1993), which works on quadrilateral meshes and the Loop scheme, which is triangular (Claes et al., 2001). The types of surfaces generated by these schemes differ, but the general principles of subdivision surfaces remain the same. In our research, we use the triangular loop scheme because triangular polygons are very well suited for modelling freeform surfaces, and they can be easily connected to an arbitrary configuration.

The Loop scheme, based on the three-dimensional box spline, is an approximating face-split scheme for triangular meshes, invented by Charles Loop (Joy, 1996). The resulting surfaces are C^2 everywhere except at extraordinary vertices — with a valence different from 6 — where they are C^1.

The Loop subdivision scheme also consists of two stages. In the splitting stage, a new vertex, called an edge point is added in the middle of each edge (the old vertices are called vertex points), and both the old and new vertices are connected to form 4 new triangles for each old triangle. In the smoothing stage, all vertices are averaged with their surrounding vertices. This smoothing step, together with the weights used, is visualised in figure 5.4, vertex points are denoted by V, edges points by E and surrounding points by Q. Figure 5.4(a) shows the subdivision mask for interior vertices. In this figure, the vertex has a valence of k. Loop's original choice for β is $\beta = \frac{1}{k}(\frac{5}{8} - (\frac{3}{8} + \frac{1}{4}\cos(\frac{2\pi}{k})^2)$, but other choices are possible as well. Figure 5.4(b) shows the subdivision mask for interior edges.

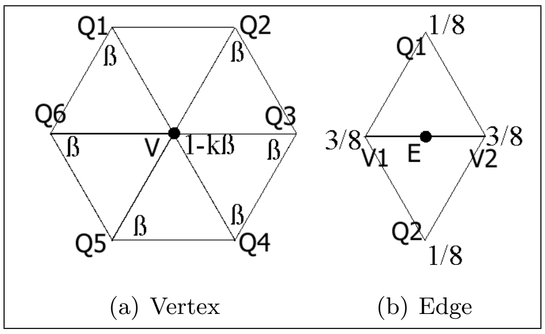


Figure 5.4: Subdivision masks for the Loop Scheme

The boundary of a subdivision surface is treated as a subdivision curve in order to avoid pulling the edge towards the interior, since only one neighbouring vertex exists.

5.4.1 Haptic rendering using Loop surfaces

During haptic rendering of polygon meshes, all of the object's faces generally need to be processed. This is no longer necessary with subdivision surfaces. The multiresolution properties of subdivision surfaces can be exploited so that only the control mesh has to be processed as a whole. In the processing stages of the following, more detailed, subdivision levels, the results of the previous test are used, thus leading to a smaller number of polygons that have to be processed. This leads to a huge increase in speed.

As mentioned earlier, a subdivision surface is defined as the limit of a series of surfaces. This gives rise to two interesting properties:

- Every face at level $n-1$ can be linked to four faces at subdivision level n.
- As can be seen from figure 5.4, the new co-ordinates of a vertex are influenced by the surrounding vertices. Using the loop subdivision scheme, most vertices have a valence of 6, so 6 vertices are needed to calculate the new co-ordinates. Generalising this for a triangle (figure 5.5), each vertex of the triangle is influenced by its 6 neighbouring triangles. Since a number of these neighbouring triangles are shared, 12 neighbouring triangles are needed to calculate the new co-ordinates of each of the triangle's vertices. Figure 5.5 shows the support of a triangle's vertices. This is the which is the region over which a point influences the shape of the limit surface. As can be seen from this figure the support in the Loop scheme is limited.

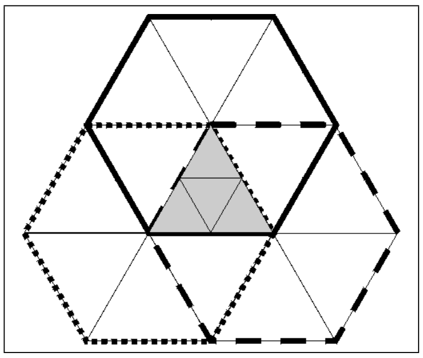


Figure 5.5: Area which influences a single triangle

The next two sections explain the two problems that need to be solved. In both cases the algorithm starts from the control mesh at level 0, leading to an accuracy up to an arbitrary subdivision level n, which contains 4^n times as much triangles as level 0 does.

5.4.2 Performing the inside-outside test

Consider a control mesh M_0, consisting of a triangles. The following steps describe the algorithm that checks wether a point p lies inside or outside the object.

1. Shoot a semi-infinite ray, starting at point p.
2. Select all the intersected faces and the face closest to the point being tested.
3. Extend this selection by including all triangles in the 1-neighbourhoods of all vertices of the intersected faces.
4. Replace all the selected faces by their subdivided children. The number of triangles is multiplied by 4. Again test all triangles in this selection for intersection with the ray.
5. Check wether the number of intersections changes from even to odd or from odd to even. This can happen because the refined meshes ”shrink” in areas where the control mesh is convex. In concave regions, the refined meshes grow outside of the control mesh. If this happened, go to step 7, otherwise proceed with step 6.
6. If the required subdivision level is not yet reached, go to step 2.
7. If the number of intersections is odd, the point lies inside the polymesh. Otherwise it lies outside the polymesh.

5.4.3 Calculating the SCP

Using the results of the previous calculations, the SCP can be calculated.

1. If in the previous algorithm the required subdivision level is found go directly to step 7.
2. Select those faces from the selected faces of the current subdivision level which are closest to the point being tested.
3. Extend the selection to faces which contain vertices in the 1-neighbourhood of all vertices in the selected faces.
4. Replace the selection with its next subdivision level.
5. Select the closest face from the previous selection.
6. If the required subdivision level is not yet reached, go to step 3.
7. Calculate the exact intersection point of the closest face. This is the SCP.

8. For each vertex of the triangle, the limit normal vector is calculated by taking the vector product of the following two vectors.

$$t_1 = \sum_{i=0}^{k-1} \cos \frac{2\pi i}{k} p_i \qquad t_2 = \sum_{i=0}^{k-1} \sin \frac{2\pi i}{k} p_i$$

The normal in the SCP is calculated by interpolating these three normals.

5.4.4 Results

Using subdivision surfaces, only a small number of triangles need to be processed. Consider again the control mesh M_0, consisting of a triangles. At level 0, a intersection tests have to be performed. If k intersections are found (k is typically a very small number), in each step these triangles and their neighbouring triangles need to be subdivided[3]. In the worst case scenario, where the neighbouring zones are al disjunct, $k * (1 + 12) * 4 = k * 52$ intersection tests have to be performed for each subdivision level. The maximum number of tests (if the test is not successful before reaching level n and all zones are always disjunct) is $a + n * k * 52$. The complexity of the algorithm is linear with respect to the subdivision level ($O(n)$). If a "normal" polymesh with the same number of polygons has to be checked, $a * 4^n$ triangles would have to be checked. The complexity of the polymesh is exponential ($O(2^n)$).

For instance, suppose a control mesh, consisting of 100 triangles, is checked until level 4 (a typical level) and 5 intersections are found. This leads to $100+4*5*52 = 1140$ checks. The equivalent polymesh consists of $a*4^4 = 25600$ triangles which leads to more than 20 times as much checks.

When using adaptive subdivision, where the fact if a triangle is subdivided depends on the surface area of the triangle (compared to the other triangles of the mesh), even a smaller number of tests is needed.

5.5 CSG Trees

A CSG model consists of a series of primitives and boolean operators, which are grouped into a tree. Most CSG implementations use the intersection ($\cap$), subtraction ($-$) and union ($\cup$) operators. An example of such a tree can be seen in figure 5.6(a). Such a tree is often written as $(A \cap B) - C$ or, assuming left association of operators, $A \cap B - C$.

[3]Please note that the subdivision process is performed in a preprocessing stage.

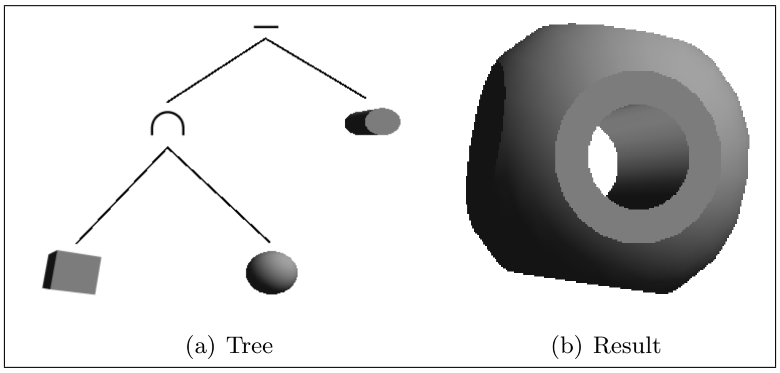


Figure 5.6: Example CSG tree

In this CSG tree, the intersection of a cube and a sphere is taken, which results in a rounded cube. Next a hole is created by subtracting a cylinder from this cube. The result of this tree is depicted in figure 5.6(b).

The usage of CSG trees has a number of advantages (Foley et al., 1990):

- Files from a number of applications can be read.
- CSG trees are accurate.
- CSG trees can easily be validated.
- CSG representations are compact.

The algorithms for interactive visual rendering of CSG models have been known for a number of years. These algorithms either use special hardware with 2 depth buffers (Steward et al., 1998) or the stencil buffer available on standard OpenGL graphics boards (Wiegand, 1996). The visual algorithm used in our implementation is based on the course notes by Blythe et al. (1999), which use the stencil buffer.

Techniques needed for haptic rendering differ radically from the graphical techniques. Graphical algorithms make use of the fact that the objects must be projected onto a viewport in such a manner that a perspective is maintained in a viewpoint. All graphical CSG algorithms make use of a variation of a ray-casting scheme. However, no haptic equivalent of such a viewport exists: haptic algorithms make objects touchable from all sides, even invisible sides.

As explained in section 5.3.2, the haptic rendering process generates a surface contact point on an object. The same principle is used in our algorithm for rendering CSG trees. In order to create a set of primitives that can be easily expanded, the algorithms do not make any assumptions on the geometries of the primitives. A primitive has to provide two methods in order to be used in a CSG tree:

1. The primitive has to check if a point lies inside or outside its surface.
2. The primitive has to calculate the SCP if the point lies inside its surface.

The algorithms will then combine the surface contact points of the subtrees.

The subtraction operator requires an extra functionality, because it has to be able to haptically render the inside of a surface:

3. The primitive has to calculate the SCP if the point lies outside its surface.

This requirement differs from requirement 2 because the techniques for rendering the outside of an object differ from the techniques to render the inside of an object. This problem is further discussed in section 5.5.3.

For most primitives the inside-outside test the calculation of an SCP if the pointer lies inside its surface are well-known and are thus not explained in this thesis. We will however elaborate on the calculation of a SCP if the pointer lies outside an object's surface and the calculations for the operation nodes.

The explanation of our algorithms, in the next sections, make use of the objects depicted in figure 5.7: a sphere and a cube that intersect each other (a cross section is shown).

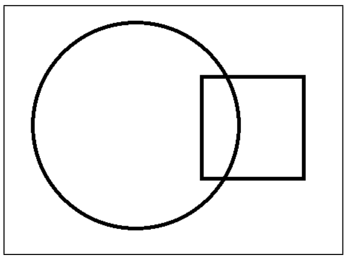


Figure 5.7: Example objects

5.5.1 Intersection

The intersection of two objects is formed by the volume that is shared by these objects. As a result, only the surfaces that are shaded black in figure 5.8 have to be felt. The rendering algorithm should ignore the greyed surfaces.

The algorithm will only calculate the SCP if the pointer is located in the volume that is defined by the black surfaces in this figure. Otherwise, the pointer is considered outside the intersection and no forces should be applied.

The inside-outside test for a intersection node should therefore test if an object is located inside the left tree and inside the right tree. Both subtrees are

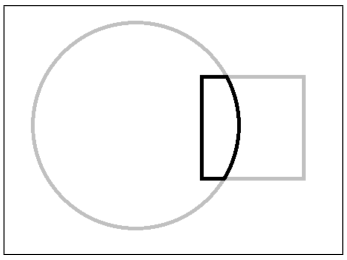


Figure 5.8: Intersection of the example objects

responsible for calculating if the pointer is located with the object represented by the subtree.

When the SCP is inside the intersection and thus passed the inside-outside test, the black surfaces are closer than the grey surfaces, so algorithm 5.1 is valid.

The intersection algorithm calculates which of the two SCPs lies closest to the pointer's position; this SCP is chosen, unless it is too far away from the previous SCP. This extra check is performed to avoid the pointer to pop through an object if to much pressure is applied by the user. The algorithm however, can only assure that this effect is avoided if the primitives do not allow the pointer to pop through themselves: an individual primitive must never allow the pointer to pop through. The current pointer position, not the previous SCP, is used as the main check, because this would cause the SCP to jitter on transitions from one subtree to the other that should feel like a smooth transition. This is not a problem in the example of figure 5.8, because the orientations of the surfaces of the sphere and cube differ; the edges in this figure are "hard edges". However, situations, exist where the transition between two objects should either not be felt or should be felt as a slight change in orientation of the surface. Experiments have shown us that a value of 50 is sufficient for the threshold to avoid pop-throughs and does not cause a jitter effect.

5.5.2 Subtraction

When subtracting an object from another object, the resulting object is defined by the volume of the latter object that is not shared with the former. The inside-outside test of a subtraction node should succeed the left subtree's inside-outside test succeeds and the right subtree's inside-outside test fails. For our example, the resulting object that is formed by subtracting a cube from a sphere is shaded black in figure 5.9.

```
GetIntersectionSCP() {
    SCP1 = LeftTree.GetSCP();
    SCP2 = RightTree.GetSCP();

    diff1 = SCP1 − PointerPosition;
    diff2 = SCP2 − PointerPosition;

    distance1 = SCP1 − PreviousSCP;
    distance2 = SCP2 − PreviousSCP;

    if diff1.length < diff2.length
        if distance1.length / distance2.length < threshold
            return SCP1;
        else
            return SCP2;
    else if distance2.length / distance1.length < threshold
        return SCP2;
    else
        return SCP1;
}
```

Algorithm 5.1: Intersection algorithm

Because the object on the right part of the tree is a primitive, as will be explained in section 5.5.5, the transition of one surface to another can never be smooth. We can therefore use the previous SCP to ensure that the pointer will not pop through the resulting object. The procedure for rendering a subtraction is given in algorithm 5.2.

The main difference between algorithms 5.1 and 5.2 is the haptic rendering of the right subtree: the inside of this subtree's surface should be rendered, since it is subtracted from the other object and the pointer does not lie inside the right subtree. The next section will discuss the problems that arise when rendering the inside surface of an object.

5.5.3 Internal rendering of an object

The rendering process for an object's outside surface is known for a large number of primitives. Figure 5.10(a) shows how the volume of a cube is divided into 6 segments in the shape of a pyramid. The base of each of these pyramidal segments is a face of the cube; if the pointer enters the segment, the SCP is calculated on this face.

The well-known algorithms for rendering objects cannot be used when

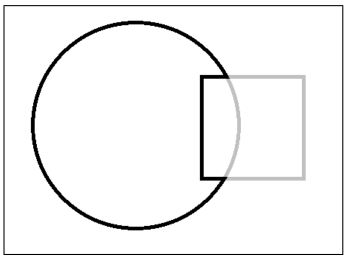


Figure 5.9: Subtracting the example objects

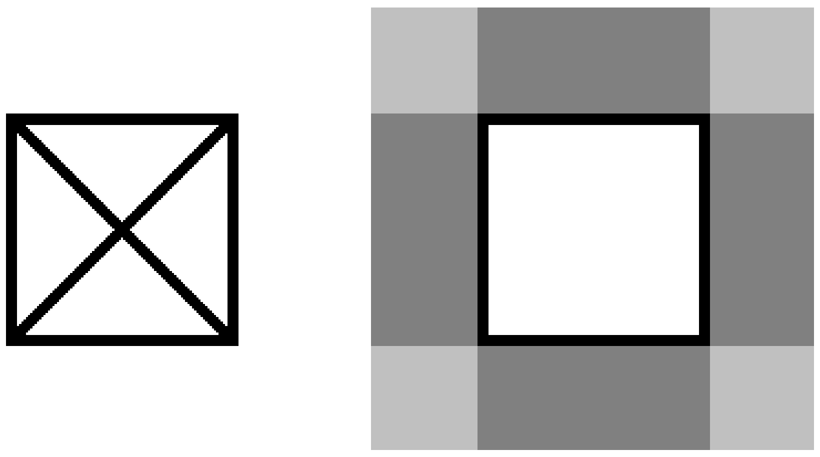
```
GetDifferenceSCP() {
    SCP1 = LeftTree.GetSCP();
    SCP2 = RightTree.GetInternalSCP();

    diff1 = SCP1 − PreviousSCP;
    diff2 = SCP2 − PreviousSCP;

    if diff1.length < diff2.length
        return SCP1;
    else
        return SCP2;
}
```

Algorithm 5.2: Subtraction algorithm

rendering the inside of the surface if the object's surface is not continuous. For instance, experiments have shown us the transition between two faces of a cube feels smooth if the segments are expanded into the space outside the cube. Therefore, when the pointer lies outside a cube, a different algorithm must be use to calculate the SCP. Contrary, the rendering algorithm for a sphere is suitable for both rendering the outside and inside surface.

The algorithm for haptic rendering of the internal surface of a cube divides the 3D world into 27 segments (figure 5.10(b) depicts a cross section of this approach in which 9 sections are shown):

- The volume within the cube. This segment is not used for the haptic rendering process since the pointer has to be located outside the cube. Figure 5.10(b) shows this segment in white.
- 6 surface segments. These segments consist of the points that can be projected onto a face of the cube by changing only 1 co-ordinate (x, y or z). Each face has one associated segment. Four of these segments are



(a) Internal segments (b) External segments

Figure 5.10: Haptic rendering of a cube (cross section)

coloured dark-grey in figure 5.10(b).

- 12 edge segments. The points that can be projected onto the edges by changing two co-ordinates make up these segments. Figure 5.10(b) displays four edge segments, coloured light-grey.
- 8 vertex segments. Points, who's x, y and z co-ordinates fall outside the range defined by the cube are projected onto the nearest vertex. These segments are not shown in figure 5.10(b).

Experiments show that, contrary to the extension of the internal algorithm, the above-mentioned procedure ensures that the edges of the cube are felt as hard edges.

In general, the inside surface of an arbitrary primitive, consisting of a number of continuous surfaces, can be rendered by dividing space into several segments: one segment for the internal volume of the primitive, one segment for each continuous surface and one segment for each transition between surfaces (both edges and vertices).

The rendering algorithm for a sphere that was mentioned earlier is thus a special case of the general approach: only two segments are defined, the internal segment and the external segment for the surface of the sphere. Using the same algorithm, one can see that the haptic rendering process for a cylinder divides space into 6 segments:

- The volume within the cylinder.

- The volume around the side of the cylinder.
- The two volumes that are the extensions of the top and bottom of the cylinder.
- The two remaining volumes that are associated with the circles that define the edges between the side and the top and bottom of the cylinder.

Algorithm 5.3 shows a general algorithm for the haptic rendering of the inside surface of an arbitrary primitive.

```
GetInternalCubeSCP() {
    for each surface segment
        if Pointer in segment
            return projection of Pointer onto surface

    for each edge segment
        if Pointer in segment
            return projection of Pointer onto edge

    for each vertex segment
        if Pointer in segment
            return vertex
}
```

Algorithm 5.3: Haptic rendering of a cube

5.5.4 Union

The union of two objects is defined as the points in space that are enclosed by either of the objects. The inside-outside test should thus succeed if the inside-outside test of one of the subtrees succeeds. The surface that should be felt for the example objects is coloured black in figure 5.11.

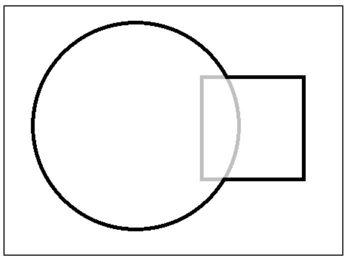


Figure 5.11: Union of the example objects

The grey surfaces in this figure define the intersection of the two objects. As already mentioned in section 5.5.1, if the pointer enters this volume, the SCP of one or both subtrees will be positioned on this intersection surface. Algorithm 5.4 takes this effect into account in order to avoid the SCP to be positioned inside the object.

```
GetUnionSCP() {
    SCP1 = LeftTree.GetSCP();
    SCP2 = RightTree.GetSCP();

    if PointerPosition in LeftTree and PointerPosition not in RightTree
        if SCP1 not in RightTree
            return SCP1;

    if PointerPosition not in LeftTree and PointerPosition in RightTree
        if SCP2 not in LeftTree
            return SCP2

    diff1 = SCP1 − PreviousSCP;
    diff2 = SCP2 − PreviousSCP;
    if diff1 .length < diff2 .length
    {
        if SCP1 not in RightTree
            return SCP1;
    }
    else if SCP2 not in LeftTree
        return SCP2;

    Candidate = PreviousSCP + PointerPosition − PreviousPointerPosition;
    SCP1 = LeftTree.GetSCP();
    SCP2 = RightTree.GetSCP();

    if ( Candidate in union object )
    {
        if Candidate in LeftTree and Candidate not in RightTree
        {
            SCP = LeftTree.GetSCP( Candidate );
            if SCP not in RightTree
                return SCP;
        }

        if Candidate not in LeftTree and Candidate in RightTree
        {
            SCP = RightTree.GetSCP( Candidate );
            if SCP not in LeftTree
```

```
                return SCP;
        }

        diff1 = SCP1 − PreviousSCP;
        diff2 = SCP2 − PreviousSCP;
        if diff1.length < diff2.length
        {
            if SCP1 not in RightTree
                return SCP1;
        }
        else if SCP2 not in LeftTree
            return SCP2;
    }

    return PreviousSCP;
}
```

Algorithm 5.4: Union algorithm

If the pointer is located in only one of the two subtrees, the algorithm uses its SCP unless this SCP is located in the other object. Otherwise the closest SCP is used if it is not located in the other object. The other SCP is not considered because this SCP is possibly located on the other side of the union object, thus allowing the pointer to pop through.

When the SCPs of both subtrees cannot be used, a heuristic should be used to calculate the exact resulting SCP. This can be done by using numerical methods which make use of knowledge of the primitives, but this is not acceptable because this information is not provided by operation nodes in the subtrees and is not required from the primitives and because these calculations would require a lot of processing time, which would cause the haptic loop to break the 0.9ms restriction.

Another, faster method is thus needed. The algorithm calculates a "candidate" SCP by moving the previous SCP in the same direction as the pointer. This candidate is then treated as a new pointer position and the same calculations are made on this candidate. If this step also fails, the algorithm stops in order to respect the timing constraints and using the previous SCP.

The proposed algorithm does not always return the exact SCP, but can give a good approximation. Experiments have shown that the errors are smaller than the resolution of the PHANToM device, which is 0.03mm and are thus hardly noticeable.

5.5.5 Tree normalisation

Most graphic rendering algorithms assume that the CSG tree is in normal form. A tree is in normal form when all intersection and subtraction operators have a left subtree that contains no union operator and a right subtree that is simply a primitive (Goldfeather et al., 1989).

A CSG tree can be converted to a normal form by repeatedly applying the following set of production rules to the tree. These production rules make use of the associative and distributive properties of boolean operations:

1. $X - (Y \cup Z) \rightarrow (X - Y) - Z$
2. $X \cap (Y \cup Z) \rightarrow (X \cap Y) \cup (X \cap Z)$
3. $X - (Y \cap Z) \rightarrow (X - Y) \cup (X - Z)$
4. $X \cap (Y \cap Z) \rightarrow (X \cap Y) \cap Z$
5. $X - (Y - Z) \rightarrow (X - Y) \cup (X \cap Z)$
6. $X \cap (Y - Z) \rightarrow (X \cap Y) - Z$
7. $(X - Y) \cap Z \rightarrow (X \cap Z) - Y$
8. $(X \cup Y) - Z \rightarrow (X - Z) \cup (Y - Z)$
9. $(X \cup Y) \cap Z \rightarrow (X \cap Z) \cup (Y \cap Z)$

Normalisation may add a number of nodes to the tree. Some of these nodes do not contribute to the final (graphical and haptic) rendering process. By calculating the bounding volume of the nodes, some nodes may be pruned. The following rules are applied for the operator nodes:

1. $Bound(A \cup B) = Bound(Bound(A) \cup (Bound(B))$
2. $Bound(A \cap B) = Bound(Bound(A) \cap (Bound(B))$
3. $Bound(A - B) = Bound(A)$

An intersection node can be pruned if the bounding boxes do not intersect. Likewise, the right subtree of a subtraction node can be pruned if the bounding boxes do not intersect.

The third requirement from section 5.5 implies that a subtraction node's right subtree is a primitive since the algorithms for the operation nodes do not guarantee that the inside of the surface is rendered properly. This is the only requirement that the haptic rendering process demands. The normalised tree can thus also be used for haptic rendering.

However, the union algorithm requires its subtree to calculate the inside-outside test and the SCP a number of times. The normalisation process brings the union operators to the root of the CSG tree, which causes the underlying tree to be calculated several times. Although it is not necessary, processor time can be saved by using an alternative normalised tree for haptic rendering. This tree is calculated by using only a subset of the production rules:

1. $X - (Y \cup Z) \to (X - Y) - Z$
2. $X - (Y \cap Z) \to (X - Y) \cup (X - Z)$
3. $X - (Y - Z) \to (X - Y) \cup (X \cap Z)$

The application of this subset leads to a tree that is smaller than the normalised tree; in worst case it is just as large as the normalised tree. Just like the normalised tree, this "subnormalised" tree can be pruned using the rules mentioned earlier.

5.5.6 Results

We implemented the CSG algorithms using the e-Touch (Novint, 2001) library. A PHANToM Premium 1.5A was used as haptic device. This device was attached to a Pentium III, 863 MHz computer with 128 MB RAM, running Windows NT. The graphical output was rendered using OpenGL and the algorithm described in (Wiegand, 1996). The different algorithms were assessed by looking at the haptic load when making the calculations.

CSG expression	No touch	Touch
Empty scene	±25%	N/A
Sphere	< 30%	±30%
Internal sphere	< 30%	±30%
Cube	< 30%	±30%
Internal cube	< 30%	±30%
Cylinder	< 30%	±30%
Internal cylinder	< 30%	±30%
Object from figure 5.6	±30%	±35%

Table 5.3: Comparison of haptic load of example CSG tree

Table 5.3 compares the haptic load for some primitives (both "normal" rendering and rendering of the inside of the primitive's surface) and the CSG tree from figure 5.6. As an extra comparison, the haptic load for an empty scene graph is also given. The second column of this table gives the haptic load when the object is not touched. This means that only the inside-outside test is performed. The third column summarises the haptic load when the pointing device touches the object, thus also requiring the calculation of a SCP.

Since an empty scene requires a haptic load of ±25% and the haptic load must always be less than 90%, as explained in section 5.3.2, ±65% of the processor time is still available for the calculation of the CSG tree's SCP. Table 5.3 shows that our algorithms stay well under the haptic load of 90%.

5.6 Integration of Force Feedback

As discussed in section 5.2, little research into general haptic environments has been performed. We believe, however, that the availability of a frame work is beneficial for research, since it allows for reuse of the results of previous research. We therefore integrated force feedback in the ICOME framework that was introduced in section 2.4. When integrating the PHANToM device in ICOME, we found that our system design should be more open and extendable through the use of the C++ inheritance mechanism. Moreover, we decided to make optimal use of the computer resources by introducing an appropriate thread scheme. Through this decision, we now have a open framework that allows for multi-modal interaction at our disposal (De Boeck et al., 2000).

5.6.1 Multithreaded design

Since the real-time aspects of an interactive virtual environment always have to be guaranteed, a multithreaded scheme has been proposed. In this paragraph, the design aspects and the necessary synchronising methods in this design will be handled in detail.

In order to allow for calculations to run simultaneously without interfering, our design consists of a number of threads, which handle the different tasks in the simulation. This design enables us to optimally make use of the hardware if multiple processors are available, as well. As shown in figure 5.12, we can define the threads as follows: one threads supports visual output. The haptic thread is the standard thread started by the GHOST haptic loop.The main thread performs the calculations on the 3D objects (such as position and orientation) and synchronies with all other threads. Provisions have been made to add threads for other functionalities without altering the existing code.

Since every thread has its own specific tasks, it is more efficient for each thread to work with its own memory-structure, rather than sharing one object list among all threads. To facilitate reuse of our code in other research projects, we have chosen to adopt the object hierarchy from ICOME and to let it co-exist with the GHOST-scene. This also allows us to include some extra features like sharing (cloning) geometries, network-IDs, textures and defining multiple scenes in a world. Furthermore, we do not depend on a force feedback API, but can use another API by only using another haptics thread.

When using different threads, each using its own scene representation, an efficient thread synchronisation scheme has to be designed. Because each particular thread synchronisation mechanism must have its own properties, different solutions are used in our framework.

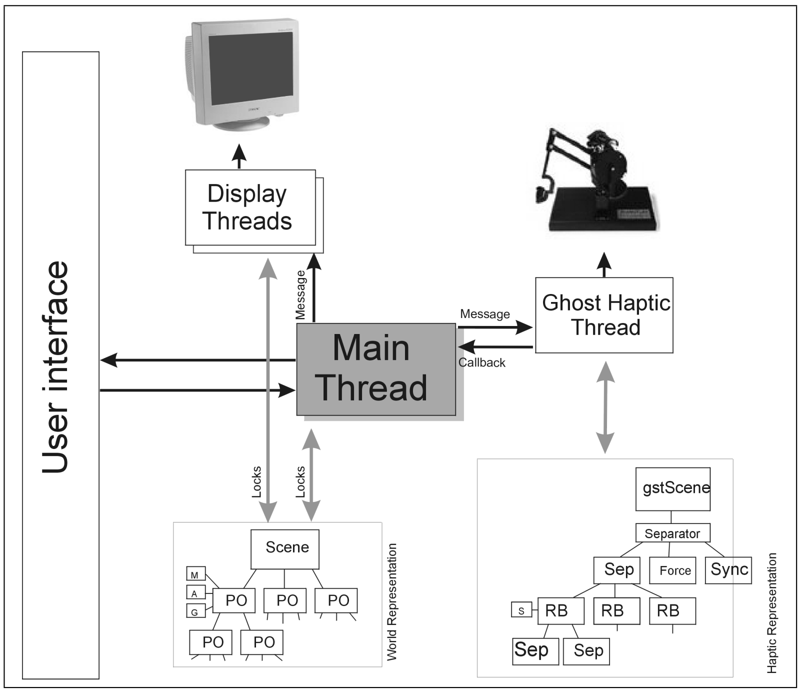


Figure 5.12: Thread scheme

The main thread communicates with the display thread through the use of messages. Therefore, we wrote a messaging class that establishes a message-pipe in which messages will be handled in full duplex using a standard first-in-first-out manner. To ensure mutual exclusion when reading and posting messages, the well-known Backery Algorithm (Silberschatz and Galvin, 1998) has been adopted in a Mutex-object. Bench-mark results, in which we tried to apply as much 'locks' and 'releases' as possible within a given time, show this algorithm to be up to 11 times faster than the standard Windows Mutex-Handle.

The communication between the main thread and the haptic thread is somewhat different. To update the main thread from changes from within the GHOST representations the standard callback methods are used. When synchronising in the other direction, a message will be sent to the haptics thread as described above. Within the GHOST scene, we have attached a synchronisation object (as can be seen in figure 5.12).

The world representation is shared by two different threads (main thread and display thread). In this particular case, all threads can read the data, while only the main thread can make changes. Therefore read and write locks are introduced in the objects so that the display thread can skip a write-locked sub-tree. In this case, the display thread first processes all other objects and resumes processing of the sub-tree at the end of the sequence. However if the object still appears to be locked, the object version known from the previous OpenGL display list is shown. Analogous, the main thread will wait before placing a write-lock on an object if a display thread holds a read-lock on it.

When an object is unlocked, update messages should be sent to the other threads. Because of the openness of our system, this functionality hasn't been centralised in the unlock code, but has been distributed over the different logical parts of the main thread. Therefore, when unlocking, objects are placed in a special-purpose queue, called the "EditedQueue" which is accessible to all update modules of the main thread.

5.6.2 Internal representation

In order to end up with an open framework in which components, functionalities or devices can be added and/or changed, some design decisions have been made. This paragraph elaborates on important design decisions that have been made within the main thread.

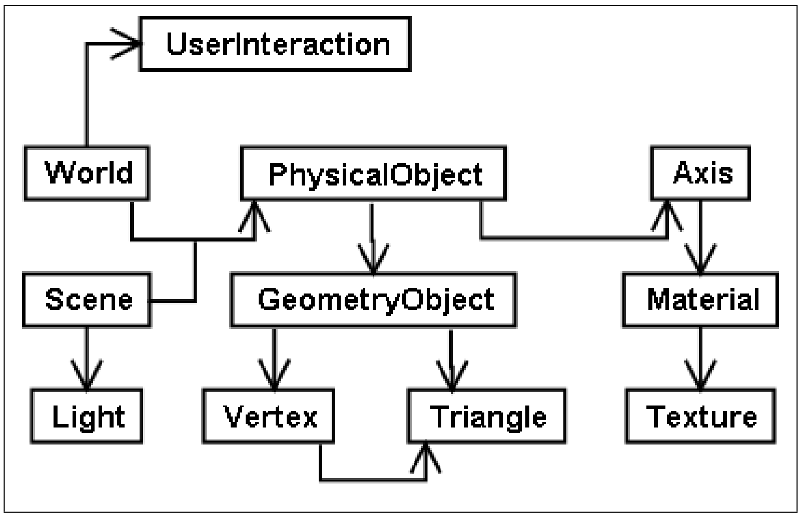


Figure 5.13: Internal world representation

In our concept, an important terminology difference must be made between worlds and scenes. A World is the collection of all objects, called physical objects, which are available in the simulation. Since those PhysicalObjects are organised in a hierarchical manner, a dynamic array, which is kept in this world, refers to the root objects of the trees. Because a World can be too complex to be manipulated as a whole, a Scene has been defined as a subset of a World. A Scene contains the subtrees of the object hierarchy that the user wants to view or interact with, so a list of these root objects is contained in a Scene object. Since the objects in the world are shared between the scenes, the modification of an object in the active scene is reflected in the other scenes as well. Switching scenes allows the user to view and manipulate just a small subset of the whole world without being hindered or disturbed by other objects. When switching between scenes, a synchronisation problem with the haptic and collision representation occurs. If we choose to hold these representations for each scene, modifying an object in the active scene implies this object to be updated in all other haptic and collision representations; even in those representations that are synchronised with inactive scenes. Since this will result in very inefficient code, only one haptic and one collision representation (namely for the active scene) exists. By consequence, a scene switch therefore results in a remake of the whole haptic representation (and other possible representations).

As in ICOME, a world contains objects, called PhysicalObjects. Those objects also refer to their hierarchical parent and children. PhysicalObjects all have a co-ordinate system and share a material with texture. The geometry of a PhysicalObject has been defined in a GeometryObject, which consists of vertices and triangles. One geometry can be shared by multiple PhysicalObjects.

Figure 5.13 depicts the classes that form the world representation. These classes implement the minimal requirements that are needed for this particular research. If later more functionality is needed, one or more classes have to be derived through the C++ inheritance mechanism. When coping with inheritance, problems arise when base classes need to instantiate derived classes. For instance in a new design, when the original base class PhysicalObject is preserved, but needs to instantiate a derived version of a GeometryObject, the original code of the PhysicalObject cannot know anything about the new geometry. We therefore adopted an object-factory mechanism: in our design, classes cannot instantiate their own member objects, but have to ask the object-factory to create those members for them. When extending the system, the newly derived classes can be simply introduced by deriving a new factory class.

5.6.3 Interaction with the user

All user interaction has been centralised in a UserInteraction component, which has been located in the main thread. By design, the UserInteraction consists of three device families:

- One or more pointer devices to control the user's cursor(s)
- A navigation device which allows the user to change his position in the world. Moving the navigation device also implies a movement of the pointer(s).
- A head device, which allows the user to look around without moving the pointer device(s) (see section 5.7.1).

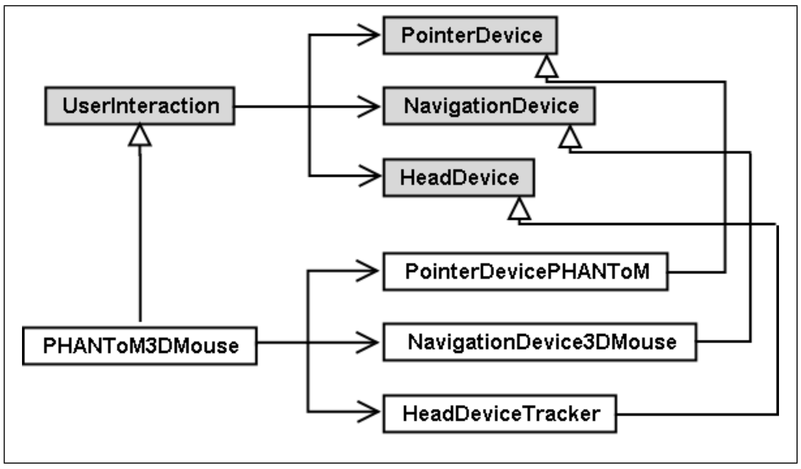


Figure 5.14: User Interaction Component

To allow the developer to freely adopt other input devices, a standard object-interface has been defined in a set of abstract classes (UserInteraction, PointerDevice, NavigationDevice and HeadDevice) (see figure 5.14). Each particular implementation for an actual device can be achieved by deriving those base classes.

In the context of our haptic and force feedback research, we have integrated the PHANToM device within this design. As can be seen from figure 5.14, the PointerDevicePHANToM class has been derived from the PointerDevice class. The new class centralises all communication and synchronisation with the haptic thread: it contains the GHOST-scene, sends the update messages to, and receives the callback functions from the haptic thread.

5.7 User Interaction in the Framework

This section discusses two different techniques that support the use in the interaction with the framework.

5.7.1 Head tracking

In order to assist the user exploring and understanding the virtual environment more thoroughly, we have developed and tested a solution in which the user's head movements are mapped to the virtual camera position. This method allows users to look at objects from a different point-of-view and enlarges the virtual workspace. This approach is comparable to the approaches used in JDCAD/JDCAD+ (Liang and Green, 1994; Green and Halliday, 1996) and HoloSketch (Deering, 1995, 1996). However, the value of this contribution is that haptic feedback, using a PHANToM device, and head tracking are combined in our experimental setup (which is not commonly found in literature).

In order to track the user's head movements, we have mounted a magnetic tracker onto a cap (see figure 5.15). We call this setup a "head-tracking device". The movements of the user's head are tracked and superimposed to the virtual camera without affecting the position of the virtual pointer, which is represented by the position (and orientation) of the PHANToM device, relative to the user.

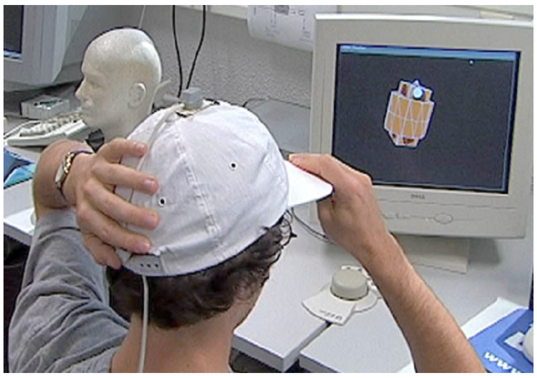


Figure 5.15: Head-tracking device

Before being applied, head movements are processed by a transfer function that limits the response range, as depicted in figure 5.16. This transfer function is a continuous symmetric function, consisting of three parts:

- A first part, which is to filter out very small movements, defines a quadratic function in the interval $[0, Knee_1]$ where $f(0) = 0$ and $f(Knee_1) = Knee_1$.

- The second part in the interval $[Knee_1, Knee_2]$ is a linear function with a slope of $Slope_1$.
- The last part of the transfer function is another linear function with $Slope_2$ typically tenths of the value of $Slope_1$. This must prevent the point of view to be completely wiped away when very large head movements or rotations are applied.

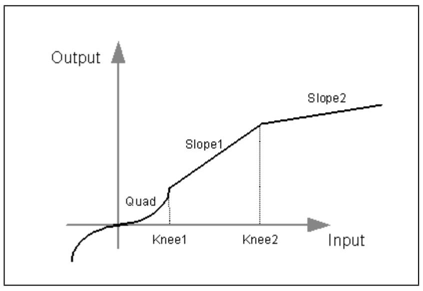


Figure 5.16: Transfer function

Tests have taught us suitable values for each of the parameters. Since translations along the Y-axis (up-down) and the X-axis (left-right) are important but hard to make while seated, those movements are amplified a bit. Rotations around the Z-axis (tilting the head) turn out to be very annoying to the user and are therefore attenuated a lot. Table 5.4 shows the actual values for each parameter. These parameters show to be satisfying for most users, however some users should perform better with a slightly different set of values.

The values of the tracker are not directly applied, but the average of the five last measurements is taken. This avoids disturbing camera movements due to jitter from the magnetic tracker.

In order to assess if the head tracking device improves the user's experience, we conducted a formal user experiment. This experiment is described in section 6.5 and in (Raymaekers et al., 2001b).

5.7.2 Menu interaction

Although menu interaction has been thoroughly explored in (traditional) VR research (Coninx et al., 1997), the implications and most effective use of 3D

	$Knee_1$	$Slope_1$	$Knee_2$	$Slope_2$
Trans X	$12cm$	1.5	$22cm$	0.5
Trans Y	$5cm$	3.0	$22cm$	0.1
Trans Z	$2cm$	1.0	$22cm$	0.1
Rot X	$5°$	0.5	$45°$	0.1
Rot Y	$5°$	0.8	$45°$	0.1
Rot Z	$5°$	0.2	$45°$	0.1

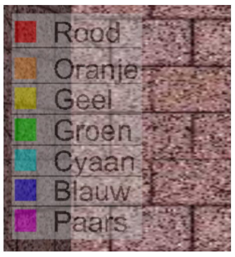
Table 5.4: Transfer function parameters

menus in a haptic context are not yet known. Different approaches to incorporate menus in a virtual environment can be identified. In JDCAD/JDCAD+ (Liang and Green, 1994; Green and Halliday, 1996) spherical and ring menus are used, while pie menus are utilised in HoloSketch (Deering, 1996). The latter approach can also be found in Alias|Wavefront's Maya (Kurtenbach et al., 2000), where a radial menu, called the Hotbox is placed around the user's mouse cursor. This is, however, a 2D solution. Lindeman et al. (1999) used hand-held windows to provide a passive feedback for manipulating 2D widgets in a 3D world with the user's finger.

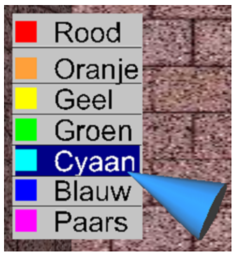
All 3D interfacing elements, such as menus, toolbars and dialogs, have to provide a common functionality. For instance, they have to support the haptic device and should allow the developer to place descriptive texts on them. We call these objects "haptic UI elements".

Another problem with 3D UI elements is the fact that they sometimes obscure the element of interest: a dialog, which asks to confirm the deletion of an object, should not be placed in front of this object. Likewise, if an object were to obscure a haptic UI element, the haptic UI element would be unusable. Therefore, a haptic UI element should be placed rather in the foreground of the virtual environment than in the background. However, it must not hinder the user in interacting with the virtual objects. Hence, we have developed haptic UI elements that can arbitrarily be positioned in the virtual environment, but are typically placed in the foreground. The haptic UI elements are semitransparent, thus not obscuring the virtual objects (Zhai et al., 1996). When the virtual pointer approaches the haptic UI element, it fades in to become opaque. As the pointer moves away, the haptic UI element fades out again. Figures 5.17(a) and 5.17(b) show a menu in its semitransparent and opaque form.

In order to determine if the pointer approaches the UI element, the distance between the pointer and the plane through the UI element is calculated. If this distance is smaller than a preset value, the pointer is projected on this



(a) Semitransparent menu



(b) Opaque menu

Figure 5.17: Menus

plane. The pointer is considered close to the UI element of the projection lies in a rectangle that is slightly larger than the UI element. In order to simplify the calculation we make use of the fact that a UI element does not lie in an arbitrary plane: a UI element is never rotated round its X or Z axis. If the UI element is used in combination with the head-tracking device(see section 5.7.1), it is possible that it is rotated around its Y axis: if a UI element that is only visible when the user turns his head, the UI element is rotated by the head orientation; this should be compensated. We therefore use a simplified plane equation by defining a base $(a, b, c), (a, b+1, c), (a + \cos\alpha, b, a + \sin\alpha)$, where (a, b, c) represents the centre point of the UI element and α is the rotation angle. This allows us to speed up the necessary calculations. The cases where $\alpha = 0$ or $\alpha = \pm 90°$ are not considered here, since these calculations are trivial.

$$\begin{vmatrix} a & b & c & 1 \\ a & b+1 & c & 1 \\ a+\cos\alpha & b & c+\sin\alpha & 1 \\ x & y & z & 1 \end{vmatrix} = 0$$

$$\Leftrightarrow x \begin{vmatrix} b & c & 1 \\ b+1 & c & 1 \\ b & c+\sin\alpha & 1 \end{vmatrix} - y \begin{vmatrix} a & c & 1 \\ a & c & 1 \\ a+\cos\alpha & c+\sin\alpha & 1 \end{vmatrix} +$$
$$z \begin{vmatrix} a & b & 1 \\ a & b+1 & 1 \\ a+\cos\alpha & b & 1 \end{vmatrix} - \begin{vmatrix} a & b & c \\ a & b+1 & c \\ a+\cos\alpha & b & c+\sin\alpha \end{vmatrix} = 0$$

$$\begin{aligned}
\Leftrightarrow & x[b(c-c-\sin\alpha) - c(b+1-b) + b\times c + b\times\sin\alpha + c + \sin\alpha - b\times c] + \\
& z[a(b+1-b) - b(a-a-\cos\alpha) + a\times b - a\times b - a - b\times\cos\alpha - \cos\alpha] + \\
& [a(b\times c + b\times\sin\alpha + c + \sin\alpha - b\times c) - \\
& b(a\times c + a\times\sin\alpha - a\times c - c\times\cos\alpha) + \\
& c(a\times b - a\times b - a - b\times\cos\alpha - \cos\alpha)] = 0
\end{aligned}$$

$$\Leftrightarrow x\times\sin\alpha - z\times\cos\alpha + c\times\cos\alpha - a\times\sin\alpha = 0 \tag{5.1}$$

The formula for the distance between the point (r, s, t) and the plane defined by equation 5.1 is:

$$\begin{aligned} & \frac{|r\times\sin\alpha - t\times\cos\alpha + c\times\cos\alpha - a\times\sin\alpha|}{\sqrt{\sin^2\alpha + \cos^2\alpha}} \\ = & |(r-a)\times\sin\alpha + (c-t)\times\cos\alpha| \end{aligned} \tag{5.2}$$

In order to find the projection onto the plane, the line through the pointer perpendicular to the plane is considered:

$$\begin{cases} (r-x)\times\cos\alpha = (z-t)\times\sin\alpha \\ y = s \end{cases} \tag{5.3}$$

The projection we seek is the intersection of equations 5.1 and 5.3. We only need the x and y co-ordinate of the projection, since the projection lies in a plane and x and z are dependent on each other. We can therefore solve equation 5.3 to z:

$$z = (r - x) \times \cot\alpha + t \tag{5.4}$$

Substituting z into equation 5.1 yields:

$$\begin{aligned}
&x \times \sin\alpha - z \times \cos\alpha + c \times \cos\alpha - a \times \sin\alpha = 0\\
\Leftrightarrow &x \times \sin\alpha - [(r - x) \times \cot\alpha + t] \times \cos\alpha + c \times \cos\alpha - a \times \sin\alpha = 0\\
\Leftrightarrow &x \times (\sin\alpha + \cot\alpha\cos\alpha) - r \times \cot\alpha \times \cos\alpha + (c - t) \times \cos\alpha - a \times \sin\alpha = 0\\
\Leftrightarrow &x \times (\frac{\sin^2\alpha + \cos^2\alpha}{\sin\alpha}) - r \times \cot\alpha \times \cos\alpha + (c - t) \times \cos\alpha - a \times \sin\alpha = 0\\
\Leftrightarrow &x = r \times \cos^2\alpha + (t - c) \times \cos\alpha\sin\alpha + a \times \sin^2\alpha
\end{aligned} \tag{5.5}$$

The co-ordinate of the projection in the dialog can be determined by x and y to the co-ordinate system of the UI element:

$$\begin{aligned}
&\begin{cases} x_{projected} = \frac{x-a}{\sin\alpha \times W/2} \\ y_{projected} = \frac{y-b}{H/2} \end{cases}\\
\Leftrightarrow &\begin{cases} x_{projected} = \frac{r \times \cos^2\alpha + (t-c) \times \cos\alpha\sin\alpha + a \times \sin^2\alpha - a}{\cos\alpha \times W/2} \\ y_{projected} = \frac{s-b}{H/2} \end{cases}\\
\Leftrightarrow &\begin{cases} x_{projected} = 2\frac{(r-a) \times \cos\alpha + (t-c) \times \sin\alpha}{W} \\ y_{projected} = 2\frac{s-b}{H} \end{cases}
\end{aligned} \tag{5.6}$$

W and H in formula 5.6 are respectively the width and the height of the UI element.

In order to use haptic UI elements in a virtual environment, we have written an abstract C++ class, which supports the above-mentioned functionalities for 3D UI interaction. When the first haptic UI element is instantiated, it makes a series of OpenGL display lists in which the alphabet is stored. All haptic UI elements can then use these display lists to display their textual contents. Likewise, the abstract class is responsible for placing a haptic constraint in the virtual environment. This haptic constraint represents the haptic UI element and calculates where the virtual pointer, representing the haptic device's position and orientation, is located with respect to the haptic UI element.

Currently, a class is derived from the abstract class, which implements a 3D menu: a 2D menu object, which can be arbitrary positioned in 3D space.

Each menu item in this 3D menu is indicated by a text, e.g. “Exit”. Optionally, a descriptive icon can be depicted in front of the menu item. In order to make the menu recognizably for the user, it employs the Windows colour scheme. Two different interaction methods are provided by the menus: “point and click” and pushing against the menu item. The first method makes use of the standard method for accessing 3D menu’s, although a 3D pointer is used in this case, while the latter is based on real life interaction with buttons and switches.

A formal user experiment was setup in order to validate the implemented user interaction techniques. This experiment and its results are discussed in section 6.6 and in (Raymaekers and Coninx, 2001).

5.8 Summary

This chapter discussed various applications of force feedback and elaborated on the techniques that are necessary for the integration of force feedback in the target domains.

Our framework and some interaction techniques that were realised in our framework were discussed in detail. The effectively of these interaction techniques were assessed in formal user experiments. The next chapter elaborates on these user experiments and gives a proof of the claimed usefulness of our interaction techniques.

Chapter 6

Evaluation of Interaction Metaphors

Contents

6.1 Introduction

The previous chapter introduced two interaction techniques that were implemented in our framework. Contrary to haptic rendering algorithms, the validity of interaction techniques cannot be calculated; they must be assessed with a user experiment. This chapter describes how user interfaces can be evaluated. Furthermore, two case studies are discussed in which the abovementioned interaction techniques were evaluated.

6.2 Evaluation

In order to evaluate our interaction techniques, we must first determine how the evaluation should be done. We therefore introduce the term "usability":

> "Usability is a *quality* of computing." (Butler et al., 2000)

> "Usability refers to the *effectiveness*, *efficiency* and *satisfaction* with which specific users can perform certain task in defined contexts." (Dillon, 2001)

Since usability is a measure for the quality of a user interface, we should evaluate the usability of our interaction techniques. The goals (Dix et al., 1998) of evaluation techniques are:

- to assess the extent of the system's functionality.
- to assess the effect of the interface on the user.
- to identify any specific problems with the system.

Different techniques exist to perform an evaluation. A number of these techniques are used in the design phase of a system. We will not elaborate on these techniques since they are based on general knowledge of the usability of a class of applications; evaluation of a new interaction metaphor should measure the effectiveness of the system, by assessing how the target users work with the system. The evaluation method used in our research is the "user experiment", which is a comparison of two or more conditions in order to assess the design of a system. It requires an experimental design (design of the test, not the system), statistical analysis and many representative users (Butler et al., 2000). A disadvantage of this method is the high number of users that are needed to overcome variance in users' skill and knowledge, thus making user experiments time-consuming.

The next sections will discuss the design of an experiment and statistical methods for analysis of the collected data.

6.3 Experimental Design

A user experiment must be carefully designed in order to obtain reliable and generalisable results and should therefore satisfy the following requirements (Dillon, 2001):

- Reliable: the results must be reproducible.
- Valid: the experiment must measure what it claims to measure.
- Representative: appropriate subjects and variables must be sampled.
- Ethical: the participants must not be harmed.

Four elements in the design of a user experiment can be identified and are explained in this section: variables, hypotheses, subjects and protocol.

6.3.1 Variables

The attributes that are manipulated and measured during the test are called variables. Dix et al. (1998) define two different sorts of variables: "dependent variables" are manipulated during the test; "independent variables" are measured. Dillon (2001) identifies a third sort of variable: "control variables" are other sources of variability whose effects we try to limit.

The independent variables represent the different conditions in the test. For example, when comparing two different interface, an independent variable is the interface that a subject is confronted with.

An example of a dependent variable is the time that a subject needs to complete the test. This variable is hypothesised to be *dependent* on the independent variables, thus on the test conditions.

Control variables are the other variables in the experiment, such as the level of expertise of a subject. If necessary, a control variable should be turned into an independent variable by adjusting the test design.

The different variables can be divided into four types (Dillon, 2001):

- Nominal variables: data that cannot be ordered (e.g. gender).
- Ordinal variables: data that can be ordered (e.g. rating scales).
- Interval variables: ordinal data that has equal intervals between adjacent units (e.g. IQ).
- Ratio variables: interval data with an absolute zero point (e.g. age).

Interval and ratio variables can further be divided into discrete variables, such as the number of errors made by the user, and continuous variables, such as completion time.

As explained in the next sessions, the definition of the variables is a very important aspect of the experimental design, since it influences the hypotheses, the division of subjects into groups and the information that is available after the experiment has been done.

6.3.2 Hypotheses

When the different variables are identified, a number of hypotheses can be defined. The hypotheses determine the analysis that is performed after the user experiment.

An example of an hypothesis is: users complete a task faster with one interface that is being tested than with the other interface that is being tested.

Often a "null hypothesis" is formulated, which states that at a given level of certainty the differences measured for a certain dependent variable occurred by chance alone, not because of the design that is being tested. Statistical analysis can calculate if a null hypothesis is valid or should be rejected.

6.3.3 Subjects

In order to get reliable results, a representative set of users must be selected. When the application that is being tested is intended for novice user, a test with experienced users cannot deliver any reliable result. Likewise, a test with an application for experienced users needs experienced users as test persons.

Next, groups must be defined, depending on the independent variables. If, for instance two interfaces are compared, half of the subjects can be confronted with one interface and the other half with the other interface.

It is also possible to let all the subjects test both interfaces. In this case the subjects also have to be divided into two groups because of transfer effects: if all subjects first test one interface and afterwards test another interface, it is likely that performance with the latter interface is better, since the subjects learned the task with the first interface. Therefore, two groups should be created: one group that is first confronted with one interface and next with another interface and one group that tests the interfaces in the other order. Of course, if the independent variable has more levels or if there is more than one independent variable, more groups have to be defined. This is called a "counterbalanced repeated measures design".

A good distribution of the subjects into the groups can limit the effect of the control variables. For instance, if the effect of gender is unlikely to influence performance with an interface, gender is not an independent variable. However, it is still possible that gender has an influence, thus being a control variable. If males and females are distributed evenly over the groups, this control variable has no or little effect.

6.3.4 Protocol

The last aspect of the experimental design that needs to be formulated is a protocol. This protocol describes the course of the experiment: the information that is given to the subjects and the questions that are being asked.

The experiment has to be explained to the users. However, if this information is oral, chances exist that not all subjects receive the same information, since it is almost impossible to always give the same explanation. On the other hand, the same written explanation can be given to all subjects, eliminating this danger that can bias the test.

It can also be important to collect subjective data: the opinions of the subjects. If this data is collected by means of a written questionnaire, the same questions can be posed with the same emphasis. It is important to ask both closed and open questions. The closed questions can be statistically analysed, while the open question can provide extra information that can explain strange results.

A goal of the experiment must be presented to the user in order to motivate the experiment, but this must not be the real goal of the experiment. If, for instance, the goal of an experiment is the look if a novel navigation method is better that the method that is commonly found in literature, this must not be told to the subject, since most people tend to pleasure the experimenter they are confronted with and (unknowingly) start with a bias. We thus need to have a goal that does not bias the subjects. In this example, the goal for the subjects can be: we are looking at how good people are in navigating in a virtual environment. The written protocol helps to avoid this problem, since it can be checked in advance. Especially when the subject answers questions in a way that is negative for the actual goal of the experiment, the written protocol must be followed. Most experimenters tend to lead the subject which leads to an invalid experiment. Again, following the written protocol avoids this problem.

In order to compensate for the learning curve of a certain task, the subjects must first be presented with a number of practice trials. The experimenters must be confident that a subject is familiar enough with the test before data is being logged. This criterium must also be defined in the protocol, thus guaranteing that all subjects start with the same confidence in the test

6.4 Statistical Analysis

Two sorts of statistics can be used to analyse the results: descriptive statistics which summarise the characteristics of a sample of data and inferential statistics which attempt to say something about a population on the basis of a sample of data (Dillon, 2001). Inferential statistics can further be categorised into methods that look for similarities (correlations) in data and methods that look for differences in data.

6.4.1 Descriptive statistics

When using descriptive statics, both the central tendency as the dispersion can be used to describe the available data.

When only nominal data (e.g. the operating system that a subject uses) is available, little calculations can be performed, since nominal data cannot be ordered. It is only possible to look at the most frequent value, called the "mode".

Ordinal data can be statistically treated by sorting the observations. The "median" is the observation in the middle of the sorted series. When the data is interval or ratio data, then the "mean", the sum of the observations (x_i) divided by the number of observations (n) can be used (equation 6.1). The big disadvantage of the mean is that it is very sensitive to outliers.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{6.1}$$

The dispersion of the data is calculated using the standard deviation (equation 6.2). When dealing with normal data (the distribution of the observations is shaped like a bell), the standard deviation indicates the width of the dispersion.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \tag{6.2}$$

Please note that the denominator of the division is $n-1$ and not n since only a sample of the total user population is observed.

Using the mean and the standard deviation, the "z score" (equation 6.3) of an observation can be calculated. This z score indicates what the chance is that the observation lies within a certain interval round the mean and can be used to identify outliers.

$$z_i = \frac{x_i - \bar{x}}{\sigma} \tag{6.3}$$

Using a statistical lookup table, one can for instance see that about 95% of all observations fall within ± 2 standard deviations of the means ($-2 \leq z \leq 2$).

Several reasons exist why data can have an outlier (or multiple outliers). It is possible that the test is wrong. In this case, the test should be redesigned and should be conducted again with other subjects. It is also possible that a user performed very bad for some reason that has nothing to do with the test design or the interface. This user must be excluded from the calculations in order to obtain valid results[1]. If a (large) number of outliers exist, chances are that they are caused by the interface design.

6.4.2 Inferential statistics: differences

The goal of a user experiment is to test if one interface is better than another. We should therefore utilise statistical techniques that can compare the observations of both interfaces and calculate which interface is better. Different tests exist, depending on the data type of a dependent variable and the sample size. We use in our research one-way and two-way ANOVA:

> Analysis of variance (ANOVA) models are versatile statistical tools for studying the relation between a response variable and one or more explanatory or predictor variables. These models do not require any assumptions about the nature of the statistical relation between the response and explanatory variables, nor do they require that the explanatory variables be quantitative. (Neter et al., 1996)

In de definition above, the response and explanatory variable are respectively the more commonly used dependent and independent variable. The mathematics behind ANOVA are explained in section 6.4.3.

One-way ANOVA can be used for experiments with one independent variable and two-way ANOVA is suitable for experiments with two independent variables.

The result of an ANOVA test is the probability that the differences in observations are caused by chance alone, not by the differences in design of the interfaces. The probability is called the "P-value" of the test. The smaller the P-value, the larger the chance that the found difference is statistically significant. Before conducting the test a confidence level must be determined; this is the greatest P-value that is still accepted as statistically significant.

[1]Ethics indicate that this information must be included when reporting about the experiment. Ideally, the calculations should be made including and excluding the outlier.

This value is important as two types of errors exist that can be made when considering probability (Dillon, 2001):

- Claiming a significant difference when there is none (type 1 error).
- Failing to claim a significant difference when there is one (type 2 error).

In HCI research a confidence level of 5% is generally accepted as significant. This convention addresses type 1 errors.

The subjective measurements that were elaborated on in section 6.3.4, are obtained using a questionnaire. Depending on the questionnaire, different statistical methods must be used. We will shortly discuss on the Chi-squared test (Dillon, 2001), since it was used to assess the questionnaire of the case study in section 6.5. The Chi-squared test is used to test if the values of an nominal are statistically significant. For instance, if a questionnaire is presented in which the user has to rate an interface (e.g. very good, good, bad, very bad), the answers will be evenly distributed if the answer would depend on chance alone. The Chi-squared test compares the observed distribution with the distribution that might would occur by chance. Using a lookup table, one can determine what the confidence level of this comparison is.

6.4.3 ANOVA

This section elaborates on the mathematics behind a one-way ANOVA model[2]. We will not explain the other ANOVA models since the are based on the same principles and since this would lead us to far. We would like to refer the interested reader to (Neter et al., 1996) and (Duchateau et al., 1998) for more information about ANOVA models.

First, some symbols need to be introduced.

Notation

This model assumes a test with one independent variable, which has r levels (for instance r interfaces for a program). Each level is denoted by an index, i. and the number of cases (measurements) for level i, is denoted by n_i. The total number of cases is:

$$n_T = \sum_{i=1}^{r} n_i \tag{6.4}$$

[2]More precisely, we will explain ANOVA model I for single-factor studies.

If the variable Y_{ij} is the jth measurement in level i, the ANOVA model can be stated as:

$$Y_{ij} = \mu_i + \varepsilon_{ij} \tag{6.5}$$

Since the model assumes that each probability distribution is normal and that each probability distribution has the same variance, we can define the expected values of ε_{ij} and thus of Y_{ij}.

$$E\{\varepsilon_{ij}\} = 0 \Rightarrow E\{Y_{ij}\} = \mu_i \tag{6.6}$$
$$\sigma^2\{Y_{ij}\} = \sigma^2\{\varepsilon_{ij}\} = \sigma^2 \tag{6.7}$$

The measurements are summarised and averaged using the following symbols:

$$Y_{i\cdot} = \sum_{j=i}^{n_i} Y_{ij} \tag{6.8}$$
$$\overline{Y}_{i\cdot} = \frac{Y_{i\cdot}}{n_i} \tag{6.9}$$
$$Y_{\cdot\cdot} = \sum_{i=1}^{r}\sum_{j=1}^{n_i} Y_{ij} \tag{6.10}$$
$$\overline{Y}_{\cdot\cdot} = \frac{Y_{\cdot\cdot}}{n_T} \tag{6.11}$$

Least square estimators

Good estimators of parameters are generally found using a least square method. We thus need to minimise the equation 6.12 in order to estimate the values of the parameters μ_i.

$$Q = \sum_i \sum_j (Y_{ij} - \mu_i)^2 \tag{6.12}$$

Since the sample mean minimises a sum of squared deviations, the least square estimator of μ_i, denoted by $\widehat{\mu}_i$ is:

$$\widehat{\mu}_i = \overline{Y}_{i\cdot} \tag{6.13}$$

Analysis of Variance

The ANOVA model is defined in terms of the total variability. This can be decomposed int de deviation of the estimated factor level mean around the overall and the deviation around the estimated factor level mean:

$$Y_{ij} - \overline{Y}_{\cdot\cdot} = \overline{Y}_{i\cdot} - \overline{Y}_{\cdot\cdot} \quad + \quad Y_{ij} - \overline{Y}_{i\cdot} \tag{6.14}$$

If this equation is squared and summed, equation 6.15 is obtained[3]:

$$\sum_i \sum_j (Y_{ij} - \overline{Y}_{\cdot\cdot})^2 = \sum_i n_i (\overline{Y}_{i\cdot} - \overline{Y}_{\cdot\cdot})^2 + \sum_i \sum_j (Y_{ij} - \overline{Y}_{i\cdot})^2 \tag{6.15}$$

We can now define the *total sum of squares* (SSTO), the *treatment sum of squares* (SSTR) and the *error sum of squares* (SSE) from equation 6.15.

$$\text{SSTO} = \sum_i \sum_j (Y_{ij} - \overline{Y}_{\cdot\cdot})^2 \tag{6.16}$$

$$\text{SSTR} = \sum_i n_i (\overline{Y}_{i\cdot} - \overline{Y}_{\cdot\cdot})^2 \tag{6.17}$$

$$\text{SSE} = \sum_i \sum_j (Y_{ij} - \overline{Y}_{i\cdot})^2 \tag{6.18}$$

$$\text{SSTO} = \text{SSTR} + \text{SSE} \tag{6.19}$$

The SSTR is a measure of the differences of between the estimated factor level means. The more these estimated factor level means differ, the larger SSTR is.

The SSTO, SSTR and SSE respectively have $n_T - 1$, $r - 1$ and $n_T - r$ degrees of freedom: the number of variables that can vary once the mean is known. Using these degrees of freedom, we can define the *treatment mean square* (MSTR) and the *error mean square* (MSE):

$$\text{MSTR} = \frac{\text{SSTR}}{r - 1} \tag{6.20}$$

$$\text{MSE} = \frac{\text{SSE}}{n_T - r} \tag{6.21}$$

$$F^* = \frac{\text{MSTR}}{\text{MSE}} \tag{6.22}$$

[3]Proof of the correctness of equation 6.15 can be found in (Neter et al., 1996).

The null hypothesis (all means are equal) holds if F^* is distributed as $F(r-1, n_T - r)$. For a level of significance α, the null hypothesis rejected if $F^* > F(1-\alpha, r-1, n_T - r)$. This value of the F-distribution can be found in a lookup table.

Intuitively, this model can be motivated as follows: if F^* is large the SSTO is mostly caused by the SSTR. This means that a high variability among the estimated factor means exists. The null hypothesis is thus not plausible.

6.4.4 Inferential statistics: similarities

Often knowledge about the correlation between two dependent variables provides important information about our design. For instance, most interfaces have a tradeoff between speed and precision: faster users make more errors. If this is the case, correlation statistics indicate a strong positive correlation between speed and the number of errors (or a strong negative correlation between completion time and the number of errors).

When confronted with interval or ration data, Pearson's Product – Moment test (also called Pearson r) is used. Suppose, the correlation between variables X and Y, both with N observations, must calculated. Pearson r is given in equation 6.23.

$$r = \frac{\sum XY - \frac{(\sum X)(\sum Y)}{N}}{\sqrt{[\sum X^2 - \frac{(\sum X)^2}{N}][\sum Y^2 - \frac{(\sum Y)^2}{N}]}} \tag{6.23}$$

The result of a correlation test is a value between -1 and 1, where -1 indicates a perfect negative correlation, 1 indicates a perfect positive correlation an 0 indicates that there is no correlation at all.

We next need to determine the degrees of freedom. In this case, the degrees of freedom is $N-2$. Knowing the confidence level and the degrees of freedom, the critical r value (r_{crit}) can be looked up. The correlation is statistically significant if $\mid r \mid > r_{crit}$.

6.5 Case Study: Head tracking

Section 5.7.1 discussed how head tracking can be used in a haptic environment. This head tracking setup has been tested in a formal user experiment which is discussed in this section.

6.5.1 Experimental setup

To test the usefulness of our solution, we have set up a user experiment in which users had to evaluate this new tool: two targeting tasks in a through-the-window VR setup, using Sensable's PHANToM device were examined. The use of the head-tracking device was compared to a condition with a fixed viewpoint. The targeting task consisted of four virtual scenes, ranging in complexity (see fig. 6.1), in which the pointer had to be moved from a fixed position, around some obstacles to a target object: a red-and-white square, positioned on a cube. Each of these scenes had to be undertaken twice in both conditions. The test persons first had the chance to practice the task in another virtual room, which only contained the target. A person, who was able to reach the target twice in less than 30 seconds, was considered as a subject with enough experience for the test.

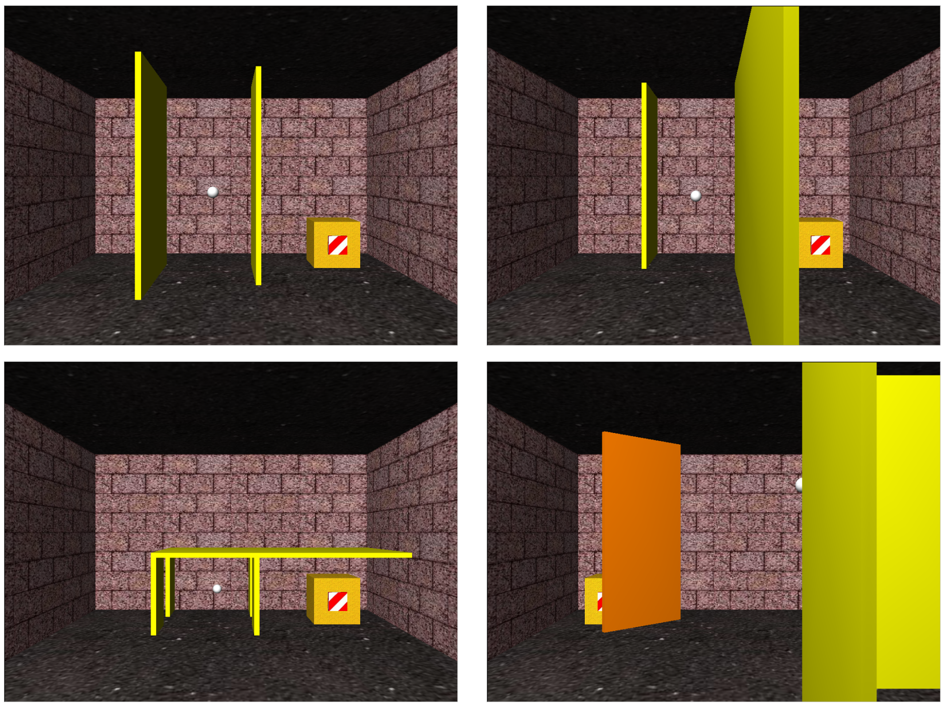


Figure 6.1: Experimental scenes

Twenty-two volunteers, twenty males and two females, all participated with their dominant hand in a counterbalanced repeated measures design. The dependent variables we measured were: elapsed time, distance covered by the virtual pointer and number of erroneous touches. After completing the test in each particular condition the test subjects were asked to answer a series of questions. When the second condition was completed, a comparing questionnaire was presented.

6.5.2 Results

After statistical analysis of the data, using one-way ANOVA over all measurements, we had to conclude that users perform better without the head-tracking device. However, if only the very last trial of each person is considered, no statistically significant difference can be found (see table 6.1). Similar results were found for the other independent variables, but these results are not so pronounced. As a first conclusion, we thus can state that people appear to be slower when they use the head-tracking. However, once the virtual room has been known well, performances in both cases show to be equal.

	All measurements	Last trial
With head-tracking device	$4719ms$	$3231ms$
Without head-tracking device	$3987ms$	$3200ms$
P-value	$< .05$	.9086

Table 6.1: Average timings

To go on with this conclusion, video recordings taken during the tests demonstrate that users behave in a more explorative manner when using head-tracking, which explains the apparent decrease in performance. This behaviour is in contrast with the more "trial-and-error" approach of the users who are not able to modify their viewpoint. Furthermore, most subjects raised the fact on the system that the head-tracking device would be of more importance when the scene becomes more complex, or when certain objects are (slightly) obscured by others. As can be seen from figure 6.1, the target is in all scenes (partially) visible. Only in the last scene the pointer is obscured in the start position. We used this kind of scenes intentionally, because otherwise the test condition without the head-tracking device would certainly be more difficult or impossible to perform, thus biasing the results.

From figure 6.2, it turns out that the users' subjective appreciation is not always confirmed by the objective measurements. In the last questionnaire, we asked the test subjects in which case they were able to reach their target in a faster manner. Almost half of the test persons reported that they were faster when they used the head-tracking device, while the other persons thought it did not make a difference, or that they were less efficient when using the head-tracking device. Using a Chi-squared test, we did not find a significant level in the answers. This result can be explained by the fact that users seem to make a subjective difference between the time needed to explore the scene and the time needed to reach the target, which the measurements in the software do not. In each trial, the position of the PHANToM was fixed for ten seconds; after a beep signal, this restriction was released and the measurements started. We believed that the subjects would benefit from the head-tracking during this time by exploring the scene and start moving the pointer to the target after the beep. However, most persons remained passive during the waiting period and did not start to explore the scene until after the beep.

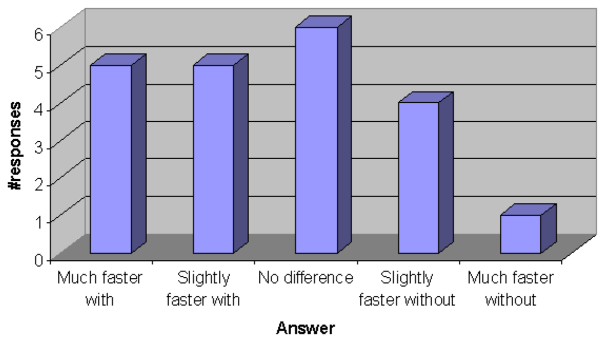


Figure 6.2: Subjective speed in both conditions

From the other answers of the comparing questionnaire, we can conclude that most persons report a better understanding of the virtual world: 63% assert having a better depth perception using the head-tracking device, while only one person found that the tracking degraded his depth perception (see figures 6.3 and 6.4). Moreover, as can be seen from figure 6.5, almost 73% of our test subjects (16 out of 22) state that working with the head-tracking device is more comfortable. No single user is indifferent with regard to the head-tracking device. The results of the Chi-squared test were significant at the 5% confidence level for the questions about depth perception and agreeability. The question about understanding of the scene is significant at the 10% level.

Besides this, we found that some test persons used the head-tracking device without noticing it: they turned their head in order to explore the scene, but afterwards reported not to have used it. One user even turned his head to look at something in his first trial, where he did not knew anything about the head-tracking device. We can thus conclude that head tracking presents a natural metaphor, even in a through-the-window VR setup. We think these results support the use of head-tracking, since using haptic feedback adds an extra

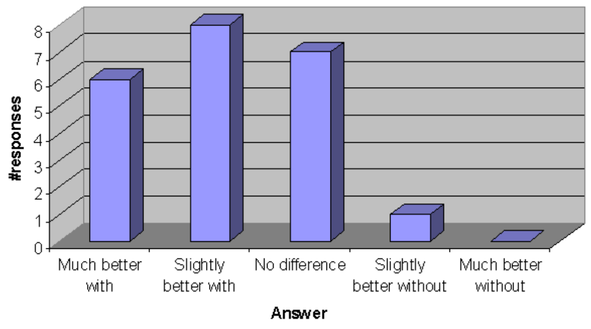


Figure 6.3: Depth perception in both conditions

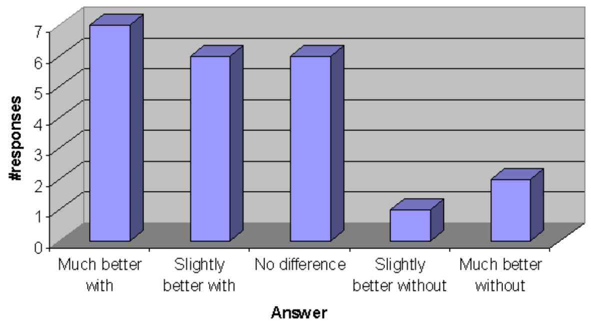


Figure 6.4: Understanding of the scene in both conditions

dimension to the user interface: a bad understanding of an object's depth, can lead to a diminished comfort due to the physical stress of bumping into objects.

6.5.3 Discussion

We can state that the head-tracking device does not result in a direct increase of performance, but rather allows the user to explore the scene in a more intuitive and natural manner and to provide him with a better spatial awareness. This is appreciated by most of our users, as we are considering e.g. object-modelling applications. We believe that the results of this experiment are transferable to other exploratory and dynamic virtual environments using similar haptic feedback. Current research has looked at the possibility to use head tracking to observe objects. This method also allows for a larger field-of-view than is made possible by a standard computer monitor. This "virtual" field-of-view could be used to add extra user interface elements in the vir-

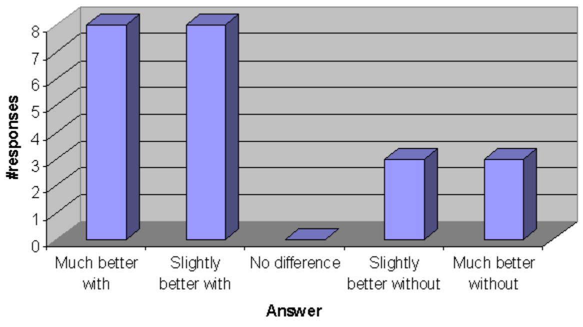


Figure 6.5: Agreeability of both conditions

tual world. For instance, a 3D menu (as discussed in section 5.7.2) could be placed just left of the normal view. If the user looked to the left, the menu would become visible and accessible. This allows user interface elements to be integrated in the virtual world, without cluttering the user's field-of-view.

6.6 Case study: Menu Interaction

This section discusses the user experiment that was setup in order to determine which kind of menu interaction (see section 5.7.2) is most suitable in haptic environments.

6.6.1 Experimental setup

We wanted to assess if the "point and click" metaphor is better than the push metaphor and if force feedback improves menu selection. A scene representing a room, containing a cube and menu, was shown to the users. The menu items indicated 7 colours; the name of each colour was preceded by a square in the same colour, so that the user could easily match the colour of the cube with a menu item. We choose to add this square instead of depicting the text in the colour it describes, because not all colours are equally readable. Figure 6.6 depicts the experimental scene; since our test persons were native Dutch speakers, the colours are indicated in Dutch. The test subjects had to indicate the current colour of the cube, which was randomly chosen, in the menu. The performance of the users when using haptic feedback, provided by a PHANToM device, was compared with a condition where no haptic feedback was present. In each condition, a test person was presented with 5 practice trials and 15 measured trials.

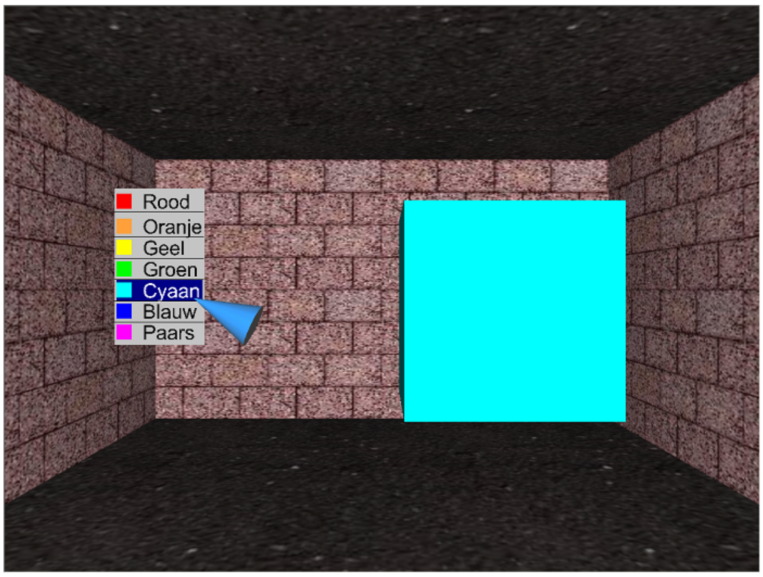


Figure 6.6: Experimental scene

Twenty-four test persons, twenty males and four females with an average age of 30, participated in a counterbalanced repeated measures design. In order to avoid negative transfer effects, the test persons where evenly distributed into two groups according to age, sex and experience with computers. One group of 12 test subjects was presented with the "point and click" interaction (called click condition), while the other group had to push against the menu items (called push condition).

The dependent variables we measured were: elapsed time (in ms), needed to select a menu item, the distance covered by the virtual pointer (in mm) and the number of erroneous selections. The elapsed time was further divided in the approach time, needed to come in the vicinity of the menu and the homing time, needed to select the item within the menu. A trial was considered complete, when the correct menu item was selected. After completing the test in each particular condition the subjects were asked to answer a series of questions. After completing the second condition, a comparing questionnaire was presented. This questionnaire presented the test persons some questions about their subjective feeling of performance and frustration.

6.6.2 Results

The results for both conditions were analysed using two-way ANOVA over all measurements (click and push conditions). This analysis reveals that the use of force feedback significantly reduces only the number of errors that were made by the test subjects. When no force was applied, the results were better for all other dependent variables, though not significantly. Table 6.2 summaries these results.

	Without force	With force	P-value
Avg. approach time	2053.7	2112.1	.86
Avg. homing time	3167.4	4076.5	.14
Avg. total time	5221.1	6188.6	.11
Avg. distance	60.7	74.8	.11
Total #errors	150	68	$<$ **.001**

Table 6.2: Statistical analysis of all results

Furthermore, the trade-off between precision and speed is confirmed by our results, since a negative correlation between the approach and homing time can be found (see table 6.3).

	Avg. approach time	Avg. homing time	Avg. distance
Avg. approach time	1.0		
Avg. homing time	-0.27	1.0	
Avg. distance	0.27	0.76	1.0
Total #errors	0.06	0.40	0.35

Table 6.3: Correlations

However if the two selection mechanisms are compared, a significant difference can be found in favour of the clicking with the stylus switch (see table 6.4). At first sight, this seems to be contra intuitive: in real-life, selecting a button, e.g. a button on a microwave oven, is done by pushing against the button, not by clicking with another device. However, in real-life people push against buttons with their fingers, not with a pen, while people are used to the "point and click" interaction when working with a computer. As an alternative, we could have chosen to use the PHANToM thimble, because it allows the user to push against objects with the index finger. However, most of our target applications require 6DOF input, which can only be provided with the encoder stylus. Furthermore, most 3D interaction benefits from having a switch on the encoder stylus (e.g. to select objects in the virtual world).

Within the click condition, no significant difference was found in the time

	Click	Push	P-value
Avg. approach time	1387.6	2778.2	$< \mathbf{.001}$
Avg. homing time	1894.9	5348.9	$< \mathbf{.001}$
Avg. total time	3282.5	8127.1	$< \mathbf{.001}$
Avg. distance	42.7	92.8	$< \mathbf{.001}$
Total #errors	79	137	$< \mathbf{.001}$

Table 6.4: Comparing the click with the push condition

needed to complete the condition with haptic feedback and the condition without haptic feedback. The approach time was slightly better when no haptic feedback was present. However, this was compensated by the better homing time in the condition with haptic feedback, leading to better over all performance in the condition with haptic feedback. Video recordings, from the user test, suggest that this difference is caused by a slight hesitation when approaching the haptic menu because the test subjects searched the haptic plane, whereas in the condition without haptic feedback, the test subjects just located the pointer in the vicinity of the menu. However, as soon as the virtual pointer hit the haptic plane, the test subjects could use this plane as a guide and worked faster and more precise. The fact that haptic feedback aids the user in being more efficient is supported by the results of the error measurements (see table 6.5). In both cases 180 (12 test persons x 15 trials) correct clicks were recorded. In the condition without haptic feedback another 63 erroneous errors were made; hence, 26% of all clicks were wrong. Haptic feedback, however reduces this figure: 8.2% of all clicks (16 out of 196) were erroneous. This result is statistically significant. We reckon that this difference is caused by test persons, who overshoot the virtual pointer when no haptic feedback is present. Furthermore, if the encoder stylus does not rest against a virtual plane, clicking on the switch will sometimes cause the stylus to move downwards before the click is registered.

	Without force	With force	P-value
Avg. approach time	1365.3	1409.8	0.82
Avg. homing time	2038.4	1751.4	0.35
Avg. total time	3403.8	3161.3	0.41
Avg. distance	42.8	$42,5$	0.94
Total #errors	63	16	$< \mathbf{.001}$

Table 6.5: Analysis of the click condition

No significant difference was found in the results of the post experiment questionnaire, although a number of test subjects in the push condition were visibly frustrated during the experiment.

6.6.3 Discussion

Our experiment shows that selection of a menu item in a haptic scene can be best performed by using a “point and click” metaphor. We believe that these results are valid for a number of selection widgets, such as toolbars and lists. In our setup, pointing can be accomplished in a reliable and fast manner. It is reliable because the menu item that can be selected lights up when the pointer is near to the item (but this is also true for the pushing mechanism). It is also fast, because the user does not have to touch the menu, lighting a menu item by coming in the vicinity is good enough. However, the menu lets the users rest their hand against a haptic plane, which helps to reduce errors. We noticed both behaviours during the test.

Of course, the haptic case still does not provide perfect selection, so usual error correction mechanisms, such as a confirmation for critical operations (e.g. clearing the virtual scene) and an undo mode for other operations, must be provided.

A problem that arose, when working with the haptic menu was the fact that it was only touchable from the front. We allowed to push trough the menu, when approaching it from the back in order to enhance the push condition. Sometimes, the test subject would get lost and ended up behind the menu. In order to push against the menu, the user must be in front of it. In this case the test person would have to go round the menu with the pointer if the back of the plane would also be touchable. We choose to keep this effect in the “point and click” condition, in order to minimise the differences between both conditions. The test persons in the push condition where happy with this feature, but the test persons in the “point and click” condition that ended up behind the menu reported that they would be more efficient if the menu would be touchable from two sides. This is an extra argument in favour of the “point and click” condition, since the use of a menu that is touchable from both sides is consistent with a haptic virtual environment. However, further research is needed to support this hypothesis.

6.7 Summary

This chapter explained the steps that are necessary to conduct a user experiment. Next two case studies were given that showed how the effectiveness of our haptic interaction techniques were assessed.

The next chapter will not only handle haptics, but will discuss how haptics and speech can be combined in order to obtain a multi-modal interface.

Chapter 7

Integrating Haptics and Speech

Contents

7.1 Introduction

The previous chapters showed how virtual environments can benefit from haptic feedback. As discussed in section 2.5, multiple modalities should be used to support the user. The incorporation of haptics is thus a first, although important, step towards a multi-modal interface for virtual environments.

As a next step, we are currently investigating the use of speech to enhance the interface. We are considering this kind of input because our users already make use of two-handed input: the PHANToM device for manipulating the virtual environment and a space mouse for navigating in the virtual environment. Furthermore, using speech to indicate what one wants to perform is a natural means because we are used to speak to other people in order to communicate what we want. However, when designing a speech interface, one must avoid the user thinking that the computer understands natural conversations, since this could frustrate the user if the computer for some (for the user) unknown reason does not understand a command(Shneiderman and Maes, 1997).

This chapter elaborates on the use of speech interfaces — speech recognition technology is not discussed, since this falls outside the scope of our research — and discusses a first test we are conducting to assess the combination of haptic feedback and speech recognition.

7.2 Speech Interfaces

The use of speech has a number of benefits (Larson, 2001):

- speech is natural
- speech enables new uses
- speech is convenient

Speech is a natural means since people constantly use speech for communication. Furthermore it enables new ways to interact because it can be used even if the user's hands are operating input devices. Therefore, speech has been used in a wide variety of applications. For example, Raman (1996) implemented a speech interface for the editor "Emacs".

However, several problems can arise when using a speech interface. Experiments have for instance shown that the visual interface can not just be translated into the speech interface (Yankelovich et al., 1995), as is also the case with the haptic interface. Larson (2001) reports that a mismatch between the users and speech interfaces exists, since the limitation of the users' memory limits the size of a verbal menu.

Another problem is the user frustration when the speech recognition is not adequate (Fischer, 1999). When a user has to repeat a formulation, his voice changes slightly. This effect is stronger when the user is emotional (e.g. frustrated). Since most speech recognition engines need to be trained, this effect makes the correct recognition of the user's utterances even more difficult for the computer, which can lead to a further frustration of the user.

User frustration must thus be avoided by careful design of the speech dialogs. Different types of speech recognition exist (Shneiderman, 1998). These types each have their own advantages and limitations that influence the possible application areas. In virtual environments research, it makes sense to use command control in order to let the user give commands. This has as advantage that only a limited dictionary is necessary, but its use can be "dangerous" since a command can be misunderstood (e.g. "clear" instead of "save") (Dix et al., 1998).

An application that makes use of speech recognition should therefore always provide adequate feedback to the user and should ask for confirmation when a potentially dangerous command is given. Furthermore, since a speech interface has no visibility of the possible commands, it should provide a limited, consequent vocabulary and a means to ask for the set of commands that are possible at that moment. If the feedback is provided by means of synthesised speech, this synthesised voice must be clear.

The next section will discuss a few example speech interfaces for virtual environments. Next, our proof-of-concept application, we created in order to assess the usefulness of a speech interface for a haptic environment, is elaborated on. Finally, some directions for future research are given.

7.3 Speech Interfaces for Virtual Environments

A number of virtual environments have been enhanced with a speech interface. The users can interact with these environments by means of different metaphors (McGlashan, 1997):

Proxy: the user controls different agents and interacts with the world through them.

Divinity: the user controls the world directly (as a "god").

Telekinesis: the objects in the world are dialog partners and are communicated directly with.

Communicative agent: an agent, separate from the world exists, which interacts with the user and carries out the user's commands.

The choice of metaphor determines the language that is being used by the users. Another important aspect in designing a speech interface for a virtual environment is the language understanding. When communicating, we expect that our dialog partner has knowledge about the world. For instance, if a world contains a red sphere, a blue cone and a green cone, the use can refer to the red sphere by, the "sphere", the "red sphere" or the "red object". Furthermore, once the user has indicated the object of interest, followup statements (e.g. "select the red sphere, make it yellow") can be made by using anaphoric references (Everett et al., 1999).

As with other applications, one must carefully choose the functionality with the virtual environment that can be accessed. We are not used to move in the real world by using voice commands; therefore voice commands are most of the time not good for navigating in the virtual world (Everett et al., 1999): a speech interface should therefore not employ commands such as "go forward" or "turn left". However, commands that indicate transportation to a precise location can be usefull[1].

An example of a speech interface for virtual environments is the speech interface of DIVERSE (McGlashan, 1997), a multi-modal interface for the DIVE distributed virtual environment architecture (Frécon and Stenius, 1998). This speech interface employs a communicative agent which also has a graphical representation, as depicted in figure 7.1. Because of this graphical representation, the agent indicates what it thinks the objects of interest are. Furthermore, since this representation does not resemble a human, but is cartoon-like, it is easier for users to accept that the agent only understands a restricted language.

Everett et al. (1999) created a speech interface for two different military applications. In a first application, they made an interface to a 3D model of a decommissioned navy ship, which is used for fire fighting research. The 3D model is used to familiarise the fire fighters with the ship. Little information about the ship is available in the original interface. Using speech recognition, the users can ask information about their location and about different rooms in the ship.

[1]This kind of transportation is also used in the real world when a driver is available (e.g. in a taxi).

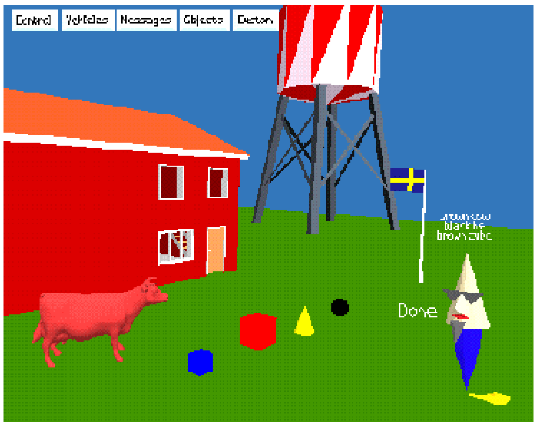


Figure 7.1: DIVERSE interface

The second speech interface is an interface for a tactical simulation playback system. The system supports a number of commands (e.g. to determine the viewpoint) and questions (e.g. about the number of ships or the heading of a helicopter).

Everett et al. detected a few usability issues in their speech interfaces. Unlike common believe, users do need training in order to work efficiently with a speech interface. Although people are already familiar with languages, they do not know the specific subset of the vocabulary and syntactic structures that can be understood by the system.

7.4 A Speech Interface for Haptic Environments

In order to assess to usefulness of a speech interface for haptic environments, we have developed a proof-of-concept application. This application provides a speech interface for a subset of commands that can be given in a haptic modelling environment.

7.4.1 Object creation

A first aspect of a modeler is the creation of objects. We allow the user to create an object by using a voice command. The user can specify the size (small, medium, large), the colour and the shape (box, cone, cylinder, sphere) of the object. Furthermore, the position of the new object relative to existing objects or to the user's cursor can be defined. In order to identify the existing objects, they have to be discriminated by their colours and shapes, as explained in section 7.3.

7.4.2 Object selection

We have created three different means to select objects. Firstly, an object can be selected by using a linear virtual hand metaphor (see also section 2.2.4): an object can be selected by touching it with the pointer and saying “select”.

The section method uses the virtual pointer metaphor (as explained in section 2.2.3): a ray is cast from the pointer; an object that is intersected with this ray can again be selected by saying “select”.

The third metaphor only uses voice commands the select an object. For instance, the user can say, “select the green cone”. This needs the same object discrimination as in the case where a new object’s position is defined relative to an existing object.

7.4.3 Object manipulation

At this moment, objects can be translated and rotated. These manipulations are initiated with a voice command. Speech is not actually used to position the objects since it is not suitable for this task. Instead, the object that is being moved follows the pointer’s movement.

Another manipulation is changing the colour of an object. A voice command can be used that makes use of a number of predefined colours.

7.4.4 Conclusions

This chapter discussed how speech can be incorporated in a haptic interface. However, further work has to be performed in order to achieve a intuitive multi-modal interface that allows to work in a haptic environment. The next chapter will discuss such future research.

Chapter 8

Directions for Future Research

8.1 Improvement of Haptic Interactions

8.1.1 Interfaces

Not much is known about haptic interfaces, such as dialogs and toolbars. Section 5.7.2 treated haptic menus, but other haptic interactions have to be investigated as well. At this moment, not much is known about dialog interaction in haptic environments. For instance, it is not known which widgets are more suitable than others. Furthermore, the way that the haptic device interacts with the widgets is also not known.

Since most haptic devices support 6 DoF, other interactions that make use of 3D space are also possible. Therefore, haptic 3D interactions and haptic hybrid 2D/3D interactions have to be weighed against each other.

At present, haptic interfaces have to be programmed from scratch. No general haptic UI development tools are available. Such a toolkit would have to take the above-mentioned issues into account. Thus, much research has to be performed to develop such a tool.

8.1.2 Object manipulations

Chapter 5 introduced two different object representations, subdivision surfaces and CSG trees, that were haptified. These object representation are suitable for object modelling. In a haptic environment, this modelling process should also be felt. Thus, modelling algorithms should be adapted to provide haptic feedback that can support the user in the modelling process.

Other object representations should be defined as well. An example representation is an implicit surface: a surface that is defined by mathematical surfaces (e.g. waves on a virtual ocean). Furthermore, object representations for soft bodies should be defined, since the object representations discussed in this thesis are rigid body representations.

8.1.3 Multiple contact point interaction

Most commercial force feedback devices only support one contact point with the virtual environment. The majority of the haptic SDKs therefore only support a limited number (mostly just one) of contact points. Some recent devices, such as haptic gloves, support multiple contact point interaction. This means that haptic environment developers either have to incorporate the specific SDKs for these devices or have to wait for the general SDKs to be adapted.

8.2 Multi-modal Interfaces

The proof-of-concept application of section 7.4 uses at this moment a set of predefined commands for colours, shapes and sizes. Provisions have to be found that let the user define custom colours sizes and shapes. This has to be realised with a limited number of voice commands in order to limit the user's mental load. Voice control is not suitable to chose from a large set of possibilities. Therefore a multi-modal interface must be developed where haptics and speech are combined to control the virtual world. An example of such a "haptics–speech" interaction is the virtual hand metaphor of section 7.4.1.

Other object manipulations can benefit from speech recognition. Tools such as free-form deformation (see also section 2.4.3) frequently need state transitions. Force feedback devices typically have a small number of buttons (one or even none). A speech interface can release this restriction.

Speech can also be used in combination with the menus that were discussed in section 5.7.2. The menu can be used to give the user a list of possible commands, which can be selected using speech or with the pointer. Especially, when submenus are needed, this multi-modal setup can be useful: the user can use the pointer to activate the menu structure and can use speech to open submenu and select menu items. A (sub)menu can be deactivated by receding the pointer from the menu and continuing with the object interaction. This way, the submenu does not have to be closed with an explicit command, but is implicitly closed when the user focuses on another object of interest.

Chapter 9

Conclusions

In this thesis, we investigated new interaction techniques for virtual environments by utilising multi-modal interfaces. More precisely, we concentrated on haptic feedback.

First, virtual environments and manipulation techniques in virtual environments were introduced. Using our 3D object modelling application, ICOME, we showed that a multi-modal interface is required to work in a virtual environment in an intuitive manner. This idea is supported by related research as explained in chapter 2.

Since touch is a very important modality, we investigated the use of touch in computer applications. First the use of passive touch was looked at. We showed the usefulness of tangible user interfaces in a number of domains and investigated the domain of cut-out animations. This TUI enables animators to position animation cells in a cut-out animation in the way they are used to manipulate animation cells.

Next, haptic feedback was introduced, together with some devices that support haptic feedback. Special attention was paid to the PHANToM device which is used in our research. We explained how haptic feedback can be used in WIMP interfaces and demonstrated this with a special purpose application for the creation of curve drawings. The haptic feedback in this application facilitates the creation and the editing of curve drawings.

The usage of haptic feedback devices can make interaction with a virtual environment more intuitive. In Chapter 5, a number of application areas for virtual environments, where haptics have successfully been used, were shown. This chapter also elaborated on the haptic rendering of 3D objects and showed that different object representations are necessary in order to advance the domain of haptic environments. We have created two such haptic object representations: subdivisions surfaces and CSG trees.

Furthermore, we showed how haptic feedback is integrated in the ICOME framework. Two interaction techniques with this framework were expounded. The problem that users have difficulties to explore and understand the virtual world was solved by using head tracking in combination with haptic feedback. In order to enable the users to give command without having to resort to mouse and keyboard, a framework for haptic UI elements were designed. As a case study, a haptic menu was implemented.

The usefulness of these interaction techniques was assessed using formal user tests. The test persons had to perform a task in different situations and had to complete a questionnaire. The users' performance and the results of the questionnaires were statistically analysed. From this analysis, we proved the usefulness of our interaction techniques.

We have started to investigate another modality: speech input. The use of a speech interface in a haptic environment was investigated. This interface showed the feasibility of voice commands in a haptic environment.

Further research has to be performed in the domains of haptic feedback and speech input for virtual environments. The issues that arose in our research are listed in the previous chapter.

We believe that the research, treated in this thesis, has laid a basis for the development of an intuitive multi-modal interface for virtual environments in general and modelling applications in particular.

Bibliography

Acosta, Eric and Temkin, Bharti (2001), *Scene Complexity: A measure for real-time stable haptic applications*, in Proceedings of the sixth PHANToM Users Group Workshop, Aspen, CO, USA

Anderson, David, Frankel, James L., Marks, Joe, Agarwala, Aseem, Beardsley, Paul et al. (2000), *Tangible Interaction + Graphical Interpretation: A New Approach to 3D Modeling*, in Proceedings of the SIGGRAPH 2000 annual conference on Computer graphics, New Orleans, LA, USA, pp. 393–402

Anderson, Tom (1997), *FLIGHT — A 3D Human-Computer Interface and Application Development Environment*, in Proceedings of the Second PHANToM Users Group Workshop, Dedham, MA, USA

Anderson, Tom, Breckenridge, Arthurine and Davidson, George (1999), *FBG: A Graphical and Haptic User Interface For Creating Graphical, Haptic User Interfaces*, in Proceedings of the Fourth PHANToM Users Group Workshop, Dedham, MA, USA

Anderson, Tom and Brown, Nick (2001), *The ActivePolygon Polygonal Algorithm for Haptic Force Generation*, in Proceedings of the sixth PHANToM Users Group Workshop, Aspen, CO, USA

Anttilla, Tommi (1998), *A haptic rendering system for virtual handheld electronic products*, VTT Publications 347, Technical Research Centre of Finland

Arsenault, Roland and Ware, Colin (2000), *Eye-Hand Co-ordination with Force Feedback*, in Proceedings of CHI 2000, Den Haag, NL, pp. 408–414

Avila, Ricardo S. (1999a), *Haptic Rendering*, chap. Volume Haptics, no. 38 in Course Notes for SIGGRAPH '99, ACM

Avila, Ricardo S. (1999b), *Haptic Rendering*, chap. Assembly and Path Planning, no. 38 in Course Notes for SIGGRAPH '99, ACM

Azouz, Naoufel and Payandeh, Shahram (2001), *A Fast Finite Element Modelling Tool for Surgical Simulation*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 7–10

Balaniuk, Remis and Costa, Ivan F. (2000), *LEM — An approach for physiaclly based soft tissue simulation suitable for haptic interaction*, in Proceedings of the fifth PHANToM Users Group Workshop, Aspen, CO, USA

Bartz, Dirk and Gürvit, Özlem (2000), *Haptic Navigation in Volumetric Datasets*, in Proceedings of the 2nd PHANToM Users Reserach Symposium 2000, vol. 8 of *Selected Readings in Vision and Graphics*, Zurich, CH, pp. 43–47

Baudel, Thomas (1994), *A Mark-Based Interaction Paradigm for Free-Hand Drawing*, in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST), Marina del Rey, CA, USA, pp. 185–192

Bier, Eric A., Stone, Maureen C., Pier, Ken, Buxton, William and DeRose, Tony D. (1993), *Toolglasses and Magic lenses: The See-Through Interface*, in Proceedings of the SIGGRAPH 1993 annual conference on Computer Graphics, Anaheim, CA, USA, pp. 73–80

Blythe, David, Grantham, Brad, McReynolds, Tom and Nelson, Scott R. (1999), *Advanced Graphics Programming Techniques Using OpenGL*, no. 29 in Course Notes for SIGGRAPH '99, ACM

Brenner, W., Mitic, S., Vujanic, A. and Popovic, G. (2001), *Micro-Actuation Principles for High-resolution Graphic Tactile Displays*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 93–98

Bryson, Steve (1994), *Developing Advanced Virtual Reality Applications*, chap. Approaches to the Successful Design and Implementation of VR Applications, no. 2 in Course Notes for SIGGRAPH '94, ACM

Burdea, Grigore C. (1996), *Force and Touch Feedback for Virtual Reality*, John Wiley & Sons

Butler, Keith A., Jacob, Robert J.K. and Preece, Jennifer (2000), *Human-Computer Interaction: Introduction and Overview*, no. 1 in Tutorial Notes of CHI '00, ACM

Catmull, E. and Clark, J. (1978), *Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes*, CAD, vol. 10(6), pp. 350–355

Chao, Dennis (2001), *Doom as an Interface for Process Management*, in Proceedings of CHI 2001, Seattle, WA, USA, pp. 152–157

Chen, C., Czerwinski, M. and Macredie, R. (1998), *Human Factors in virtual Environments*, Virtual Reality, vol. 3(4), pp. 223–225

Chen, Elaine (1999), *Six Degree-Of-Freedom Haptic System for Desktop Virtual Prototyping Applications*, in Proceedings of First International Workshop on Virtual Reality and Prototyping, Laval, FR, pp. 97–106

Cholewiak, Roger W., Collins, May A. and Brill, J. Christopher (2001), *Spatial Factors in Vibrotactile Pattern Perception*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 41–47

Claes, Johan, Coninx, Karin and Vansichem, Gert (2000), *Using image recognition to facilitate intuitive input*, in Proceedings of the Third International Workshop on New Approaches to High-tech:Testing and Computer Simulations in Science and Engineering, St. Petersburg, RU, pp. 233–238

Claes, Johan, Ramaekers, Marc and Van Reeth, Frank (2001), *Providing Local Interpolation, Tension and Normal Control in the Manipulation of Loop Subdivision Surfaces*, in Proceedings of Computer Graphics International 2001, Hong-Kong, pp. 299–305

Cohen, Abe and Chen, Elaine (1999), *Six Degree-Of-Freedom Haptic System as a Desktop Virtual Prototyping Interface*, in Proceedings of the ASME Winter Anuual Meeting, Dynamics Sytems and Control, vol. 67 of *DCS*, Nashville, TE, USA, pp. 401–402

Cohen, J., Markosian, L., Zeleznik, R., J., Hughes. and Barzel, R. (1999), *An Interface for Sketching 3D Curves*, in Proceedings of the 1999 ACM Symposium on interactive 3D Graphics, Atlanta, GE, USA, pp. 17–21

Coninx, Karin (1997), *Hybrid 2D/3D Human-Computer Interaction Techniques in Immersive Virtual Modelling Environments*, Ph.D. thesis, Limburg University Centre, Diepenbeek, BE

Coninx, Karin, Raymaekers, Chris and De Weyer, Tom (1999), *Uniform Interaction for Immersive and Non-Immersive 3D Object Modelling*, in Proceedings of Human-Computer Interaction International, vol. 2, Munich, DE, pp. 271–275

Coninx, Karin, Van Reeth, Frank and Flerackers, Eddy (1997), *A Hybrid 2D/3D User Interface for Immersive Object Modeling*, in Proceedings of

Computer Graphics International '97, Hasselt and Diepenbeek, BE, pp. 47–55

Conway, Matthew, Audia, Steve, Burnette, Tommy, Cosgrove, Dennis, Christiansen, Kevin et al. (2000), *Alice: Lessons Learning from Building a 3D System for Novices*, in Proceedings of CHI 2000, Den Haag, NL, pp. 486–493

Crossan, Andrew, Brewster, Stephen, Reid, Stuart and Mellor, Dominic (2000), *Multimodal feedback Cues to Aid Veterinary Training Simulations*, in Proceedings of the Haptic Human-Computer Interaction Workshop, Glasgow, UK

Davis, N.J., Morgan, M.D. and Wing, A.M. (2001), *The Grasp-span Weight illusion*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 22–25

De, Suvrana and Srinivasan, Mandayam A. (1998), *Rapid Rendering of "Tool-Tissue" Interaction in Surgical Simulations; Thin Walled Membrane Models*, in Proceedings of the Third PHANToM Users Group Workshop, Dedham, MA, USA

De Boeck, Joan, Raymaekers, Chris and Coninx, Karin (2001), *Expanding the Haptic Experience by Using the PHANToM Device to Drive a Camera Metaphor*, in Proceedings of the sixth PHANToM Users Group Workshop, Aspen, CO, USA

De Boeck, Joan, Raymaekers, Chris and Van Reeth, Frank (2000), *Introducing Touch in a Rigid Body Simulation Environment: a Design Overview*, in Proceedings of the 2nd PHANToM Users Reserach Symposium 2000, vol. 8 of *Selected Readings in Vision and Graphics*, Zurich, CH, pp. 1–8

De Weyer, Tom, Raymaekers, Chris and Coninx, Karin (1999), *Immersive 3D Object Modelling in ICOME*, in Proceedings of WSCG 99, Plzen, CZ, pp. SP/126–SP/133

Deering, M.F. (1995), *HoloSketch: A Virtual Reality Sketching/Animation Tool*, ACM Transactions on Computer-Human Interaction, vol. 2(2), pp. 220–238

Deering, M.F. (1996), *The HoloSketch VR Sketching System*, Communications of the ACM, vol. 39(5), pp. 54–61

Dennerlein, Jack Tigh, Martin, David B. and Hasser, Christopher (2000), *Force-Feedback Improves Performance For Steering and Combined Steering-Trageting Tasks*, in Proceedings of CHI 2000, Den Haag, NL, pp. 423–429

DeRose, Tony, Kass, Michael and Truong, Tien (1999), *Subdivision Surfaces in Character Animation*, in Proceedings of the SIGGRAPH 1998 annual conference on Computer Graphics, Orlando, FL, USA, pp. 85–94

Deutsch, Judith E., Latonio, Jason, Burdea, Grigore and Boian, Rares (2001), *Rehabilitation of Musculoskeletal Injuries Using the Rutgers Ankle Haptic Interface: Three Case Reports*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 93–98

Dillon, Andrew (2001), *Test Design and Statistical Data Analysis for Usability Evaluation*, no. 20 in Tutorial Notes of CHI '01, ACM

Dinh, Huong Q., Walker, Neff and Hodges, Larry F. (1999), *Evaluating the Importance of Multi-sensory Input on Memory and the Sense of Presence in Virtual Envrionments*, in Proceedings of IEEE Virtual Reality '99, Houston, TE, USA, pp. 222–228

Dix, Alan, Finlay, Janet, Abowd, Gregory and Beale, Russell (1998), *Human-Computer Interaction*, Prentice Hall

dos Santos Machado, Liliane, de Moraes, Ronei Marcos and Zuffo, Marcelo Knorich (2000), *Fuzzy Rule-Based Evaluation for a Haptic and Stero Simultor for Bone Marrow Harvest for Transplant*, in Proceedings of the fifth PHANToM Users Group Workshop, Aspen, CO, USA

Duchateau, Luc, Janssen, Paul and Rowlands, John (1998), *Linear Mixed Models*, International Livestock Research Institute

Everett, S.S., Wauchope, K. and Pérez Quiñones, M.A. (1999), *Creating Natural Languge Interfaces to VR Systems*, Virtual Reality, vol. 4(2), pp. 103–113

Evestedt, Daniel and McLaughin, John (2001), *Mutual Calibration of a Co-Located Haptics Device and Stereoscopic Display*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 155–159

Fischer, Kerstin (1999), *Repeats, Reformulations and Emotional Speech: Evidence for the Design of Human-Computer Speech Interfaces*, in Proceedings of Human-Computer Interaction International, vol. 1, Munich, DE, pp. 560–565

Fitzmaurice, George, Baudel, Thomas, Kurtenbach, Gordon and Buxton, Bill (1997), *A GUI Paradigm Using Tablets, Two-hands and Transparency*, in Proceedings of CHI 1997, vol. Extended Abstracts, Atlanta, GA, USA, pp. 212–213

Fitzmaurice, George W. and Buxton, Bill (1997), *An empirical Evalution of Graspable User Interfaces: towards specialized, space-multiplexed input*, in Proceedings of CHI 1997, Atlanta, GA, USA, pp. 43–50

Fitzmaurice, George W., Ishii, Hiroshi and Buxton, William (1995), *Bricks: Laying the Foundations for Graspable User interfaces*, in Proceedings of CHI 1995, Denver, CO, USA, pp. 442–449

Fjeld, Morten, Voorhorst, Fred, Bichsel, Martin and Krueger, Helmut (1999), *Exploring Brick-Based Camera Control*, in Proceedings of Human-Computer Interaction International, vol. 2, Munich, DE, pp. 1060–1064

Fjeld, Morten, Voorhorst, Fred, Bichsel, Martin, Krueger, Helmut and Rauterberg, Matthias (2000), *Navigation Methods for an Augmented Reality System*, in Proceedings of CHI 2000, vol. Extended Abstracts, The Hague, NL, pp. 8–9

Flerackers, Chris, Earnshaw, Ray, Vansichem, Gert, Van Reeth, Frank and Alsema, Frank (2001), *Creating Broadcasted Interactive Drama in a Networked Virtual Environment*, IEEE Computer Graphics and Applications, vol. 21(1), pp. 56–60

Flerackers, Chris, Raymaekers, Chris, Vansichem, Gert and Van Reeth, Frank (2000), *Generating Interactive Television Programs in the PANIVE architecture*, in Proceedings of Digital Content Creation Conference, Bradford, UK, pp. 368–375

Foley, James D., van Dam, Andries, Feiner, Steven K. and Hughes, John F. (1990), *Computer graphics: principles and practice*, Systems Programming Series, Addison-Wesley

Frécon, Emmanuel and Stenius, Mårten (1998), *DIVE: A scalable network architecture for distirbuted virtual environments*, Distributed Systems Engineering Journal, vol. 5(3), pp. 91–100

Girone, M., Burdea, G., Bouzit, M., Popescu, V. and Deutsch, J. (2001), *A Stewart Platform-Based System for Ankle Telerehabilitation*, Autonomous Robots, vol. 10, pp. 203–212

Goldfeather, Jack, Molnar, Steven, Turk, Greg and Fuchs, Henry (1989), *Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning*, IEEE Computer Graphics and Applications, vol. 9(3), pp. 20–27

Gorman, Paul J., Lieser, J.D., Murray, W.B., Haluck, Randy S. and Krummel, Thomas M. (1998), *Assesment and Validation of a Force Feedback Virtual Reality Based Surgical Simulator*, in Proceedings of the Third PHANToM Users Group Workshop, Dedham, MA, USA

Grange, Sébastien, Conti, François, Rouiller, Patrice, Helmer, Patrick and Baur, Charles (2001), *Overview of the Delta Haptic Device*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 164–166

Green, M. and Halliday, S. (1996), *A Geometric Modeling and Animation System for Virtual Reality*, Communications of the ACM, vol. 39(5), pp. 46–53

Halstead, Mark, Kass, Michael and DeRose, Tony (1993), *Efficient, Fair Interpolation using Catmull-Clark Surfaces*, in Proceedings of the SIGGRAPH 1993 annual conference on Computer Graphics, Anaheim, CA, USA, pp. 73–80

Hinckley, Ken, Pausch, Randy, Coble, John C. and Kassell, Neal F. (1994a), *Passive Real-World Interface Props for Neurosurgical Visualization*, in Proceedings of CHI 1994, Boston, MA, USA, pp. 452–458

Hinckley, Ken, Pausch, Randy, Coble, John C. and Kassell, Neal F. (1994b), *Survey of Design Issues in Spatical Input*, in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST), Marina del Rey, CA, USA, pp. 213–222

Hoffman, H.G., Hollander, A., Schroder, K., Rousseau, S. and Furness, T. (1998), *Physically Touching and Tasting Virtual Objects Enhances the Realism of Virtual Experiences*, Virtual Reality, vol. 3(4), pp. 226–234

Igarashi, Takeo, Matsuoka, Satoshi and Tanaka, Hidehiko (1999), *Teddy: A Sketching Interface for 3D Freeform Design*, in Proceedings of the SIGGRAPH 1999 annual conference on Computer graphics, Los Angeles, CA, USA

Ishii, Hiroshi and Ullmer, Brygg (1997), *Tangible Bits: towards Seamless Interfaces between People, Bits and Atoms*, in Proceedings of CHI 1997, Atlanta, GA, USA, pp. 234–241

Joy, Ken (1996), *On-Line Geometric Modeling Notes*, web pages available at http://muldoon.cipic.ucdavis.edu/CAGDNotes/. Computer Science Department, University of California

Kabeláč, Zdeněk (2000), *Rendering stiff walls with PHANToM*, in Proceedings of the 2nd PHANToM Users Reserach Symposium 2000, vol. 8 of *Selected Readings in Vision and Graphics*, Zurich, CH

Kalawski, Roy S. (1993), *The Science of Virtual Reality and Virtual Environments*, Addison-Wesley

Kälviänen, H. (1994), *Randomized Hough Transforms: New Extensions*, Ph.D. thesis, Lappeenranta University of Technology, FI

Kälviänen, H., Hirvonen, P. and Oja, E. (1995), *Houghtool: a Software Package for Hough Transform Calculation*, in Proceedings of the 9th Scandinavian Conference on Image Analysis, vol. 2, Uppsala, SW, pp. 841–848

Kurtenbach, G., Fitzmaurice, G.W., Owens, R.N. and Baudel, T. (2000), *The Hotbox: Efficient Access to a Large Number of Menu-items*, in Proceedings of CHI 2000, Den Haag, NL, pp. 231–237

Kurtenbach, Gordon, Fitzmaurice, George, Baudel, Thomas and Buxton, Bill (1997), *The Design of a A GUI Paradigm based on Tablets, Two-hands and Transparency*, in Proceedings of CHI 1997, Atlanta, GA, USA, pp. 35–42

Křenek, Aleš (2000), *Haptic Rendering of Molecular Felxibility*, in Proceedings of the 2nd PHANToM Users Reserach Symposium 2000, vol. 8 of *Selected Readings in Vision and Graphics*, Zurich, CH

Křenek, Aleš (2001), *Haptic Rendering of Molecular Conformations*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 142–145

Larson, James A. (2001), *Introduction to VoiceXML — the Markup Language for Web-Based Voice Applications*, no. 3 in Tutorial Notes of CHI '01, ACM

Lee, Jay, Ishii, Hiroshi, Duun, Blair, Su, Voctor and Ren, Sandia (2001), *GeoSCAPE: Designing a Reconstructive Tool for Field Archeological Excavation*, in Proceedings of CHI 2001, vol. Extended abstracts, Seattle, WA, USA, pp. 35–36

Lee, Jay, Su, Victor, Ren, Sandia and Ishii, Hiroshi (2000), *HandSCAPE: A Vectorizing Tape Measure for On-Site Measuring Applications*, in Proceedings of CHI 2000, Den Haag, NL, pp. 137–144

Liang, J. and Green, M. (1994), *JDCAD: A Highly Interactive 3D Modeling System*, Computers & Graphics, vol. 18(4), pp. 499–506

Lindeman, Robert W., Sibert, John L. and Kahn, James K. (1999), *Hand-Held Windows: Towards Effective 2D Interaction in Immersive Virtual Environments*, in Proceedings of IEEE Virtual Reality '99, Houston, TE, USA, pp. 205–212

Long, A. Chris, Landay, James A., Rowe, Lawrence A. and Michiels, Joseph (2000), *Visual Similarity of Pen Gestures*, in Proceedings of CHI 2000, vol. Extended Abstracts, The Hague, NL, pp. 360–367

Loureiro, Rui, Amirabdollahian, Frashid, Coote, Susan, Stokes, Emma and Harwin, William (2001), *Using Haptics Technology to Deliver Motivational Therapies in Stroke Patients: Concepts and Initial Pilot Studies*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 164–166

Mankoff, Jennifer, Hudson, Scott E. and Abowd, Gregory D. (2000), *Providing Integrated toolkit-Level Support for Ambiguity in Recognition-Based Interface*, in Proceedings of CHI 2000, vol. Extended Abstracts, The Hague, NL, pp. 368–375

Mark, William R., Randolph, Scott C., Finch, Mark, Verth, James M. Van and Taylor, Russel M. (1996), *Adding Force to Graphics Systems: Issues and Solutions*, in Proceedings of the SIGGRAPH 1996 annual conference on Computer Graphics, New Orleans, LA, USA, pp. 447–452

Massie, Thomas H. and Salisbury, J. K. (1994), *The PHANToM Haptic Interface: A Device for Probing Virtual Objects*, in Proceedings of the ASME Winter Annual Meeting, Interfaces for Virtual Enviroments and Teleoperator Systems, Chicago, IL, USA

McGee, Marilyn Rose (1999), *A haptically enhanced scrollbar: Force-Feedback as a means of reducing the problem associated with scrolling*, in Proceedings of the first PHANToM Users Reserach Symposium 2000, Heidelberg, DE

McGlashan, Scott (1997), *Speech Interfaces to Virtual Reality*, in Proceedings of The Second International Conference on the Military Applications of Synthetic Environments and Virtual Reality, Stockholm, SW

McLaughlin, J.P. and Orenstein, B.J. (1997), *Haptic Rendering of 3D Seismic Data*, in Proceedings of the Second PHANToM Users Group Workshop, Dedham, MA, USA

Miller, Timoty (1998), *Implementation Issues in Adding Force Feedback to the X Desktop*, in Proceedings of the Third PHANToM Users Group Workshop, Dedham, MA, USA

Miller, Timoty and Zeleznik, Robert (1998), *An Insidious Haptic Invasion: Adding Force Feedback to the X Desktop*, in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST), San Francisco, CA, USA, pp. 59–64

Mine, Mark R. (1996), *Working in a Virtual World: Interaction Techniques Used in the Chapel Hill Immersive Modeling Program*, Technical report 96-029, University of North Carolina

Mine, Mark R., Brooks, Frederik P. and Sequin, Carlo H. (1997), *Moving Objects in Space: Exploiiting Proprioception in Virtual-Environment Interaction*, in Proceedings of the SIGGRAPH 1997 annual conference on Computer Graphics, Los Angeles, CA, USA, pp. 19–26

Mor, Andrew B., Gibson, Sarah and Samosky, Joseph T. (1996), *Interacting with 3-Dimensional Medical Data Haptic Feedback for Surgical Simulation*, in Proceedings of the first PHANToM Users Group Workshop, Dedham, MA, USA

Neter, John, Kutner, Michael H., Nachtsheim, Christopher J. and Wasseman, William (1996), *Applied Linear Statistical Models*, 4 edn., McGraw-Hill

Novint (2001), *e-Touch Programmers Guide*

Oakley, Ian (1999), *Comparing haptic effects in a GUI*, in Proceedings of the first PHANToM Users Reserach Symposium 2000, Heidelberg, DE

Oakley, Ian, McGee, Marilyn Rose, Brewster, Stephen and Gray, Philip (2000), *Putting the feel in 'Look and Feel'*, in Proceedings of CHI 2000, Den Haag, NL, pp. 415–422

O'Toole, R., Playter, R., Blank, W., Cornelius, N., Roberts, W. et al. (1997), *A novel Virtual Reality Surgical trainer with Force Feedback: Surgeon vs. Medical Student Performance*, in Proceedings of the Second PHANToM Users Group Workshop, Dedham, MA, USA

Poupyrev, I., Weghorst, S., Billinghurst, M. and Ichikawa, T. (1998), *Egocentric Object Manipulation in Virtual Environments: Empirical Evaluation of Interaction Techniques*, in 1998 Eurographics/SIGGRAPH Workshop on Graphics Hardware, Lisbon, PT, pp. 41–52

Poupyrev, Ivan, Billinghurst, Mark, Weghorst, Suzanne and Ichikawa, Tadao (1996), *The Go-Go Interaction Technique: Non-linear Mapping for Direct*

Manipulation in VR, in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST), Seattle, WA, USA, pp. 79–80

Press, William H., Flannery, Brian P., Teukolsky, Saul A. and Vetterling, William T. (1995), *Numerical Recipies in C, The Art of Scientific Computing*, 2 edn., Cambridge University Press

Raman, T.V. (1996), *Emacspeak — A Speech Interface*, in Proceedings of CHI 1996, Vancouver, CA, pp. 66–71

Ranta, John F. and Aviles, Walter A. (1999), *The Virtual Reality Dental Training System — Simulating dental procedures for the purpose of training dental students using haptics*, in Proceedings of the Fourth PHANToM Users Group Workshop, Dedham, MA, USA

Raymaekers, Chris, Beets, Koen and Van Reeth, Frank (2001a), *Fast Haptic Rendering of Complex Objects Using Subdivision Surfaces*, in Proceedings of the sixth PHANToM Users Group Workshop, Aspen, CO, USA

Raymaekers, Chris and Coninx, Karin (2001), *Menu Interactions in a Desktop Haptic Environment*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 49–53

Raymaekers, Chris, De Boeck, Joan and Coninx, Karin (2001b), *Assessing Head-Tracking in a Desktop Haptic Environment*, in Proceedings of HCI International 2001, vol. 1, New Orleans, LA, USA, pp. 302–306

Raymaekers, Chris, De Weyer, Tom, Coninx, Karin, Van Reeth, Frank and Flerackers, Eddy (1999), *ICOME: an Immersive 3D Object Modelling Environment*, Virtual Reality, vol. 4(4), pp. 265–274

Raymaekers, Chris, Vansichem, Gert and Van Reeth, Frank (2000), *Using Haptic Feedback to Improve 2D Curve Creation with a 2.5D Pen-Like Metaphor*, in Proceedings of the 2nd PHANToM Users Reserach Symposium 2000, vol. 8 of *Selected Readings in Vision and Graphics*, Zurich, CH, pp. 27–34

Raymaekers, Chris, Vansichem, Gert and Van Reeth, Frank (2002), *Improving Sketching by Utilizing Haptic Feedback*. Accpeted for AAAI Spring Symposium on Sketch Understanding, Palo Alto, CA, USA

Reinig, Karl D. (1996), *Haptic Interaction with the Visible Human*, in Proceedings of the first PHANToM Users Group Workshop, Dedham, MA, USA

Ruspini, Diego (1999), *Haptics: From Basic Principles to Advanced Applications*, chap. Haptic Rendering, no. 38 in Course Notes for SIGGRAPH '99, ACM

Ruspini, Diego C., Kolarov, Krasimir and Khatib, Oussama (1997), *The Haptic Display of Complex Graphical Environments*, in Proceedings of the SIGGRAPH 1997 annual conference on Computer Graphics, Los Angeles, CA, USA, pp. 345–352

Sachs, Emanuel, Roberts, Andrew and Stoops, David (1991), *3-Draw; A Tool for Designing 3D Shapes*, IEEE Computer Graphics and Applications, vol. 11(6), pp. 18–25

Salisbury, J. Kenneth and Srinivasan, Manayam A. (1997), *PHANToM-Based Haptic Interaction with Virtual Objects*, IEEE Computer Graphcis & Applications, vol. 7(5), pp. 6–10

Sato, Makoto (2001), *Evolution of SPIDAR*, in Proceedings of VRIC 2001, Laval, FR, pp. 81–84

Savchenko, Vladimir (2000), *3-D Gemeotric Modeller with Haptic Feedback: Engraving Simulation*, in Proceedings of the 2nd PHANToM Users Reserach Symposium 2000, vol. 8 of *Selected Readings in Vision and Graphics*, Zurich, CH, pp. 43–47

Schmandt, Christopher (1983), *Spatial Input/Display Correspondance in a Stereoscopic Computer Graphic Work Station*, in Proceedings of the SIGGRAPH 1983 annual conference on Computer Graphics, Detroit, MI, USA, pp. 253–259

Schneider, Philip J. (1990), *Graphic Gems*, chap. An Algorithm for Automatically Fitting Digitized Curves, Academic Press, pp. 612–626

Sederberg, Thomas W. and Parry, Scott R. (1986), *Free-Form Deformation of Solid Geometric Models*, in Proceedings of the SIGGRAPH 1986 annual conference on Computer Graphics, Dallas, TE, USA, pp. 151–160

Seeger, Adam, Chen, Jun and Taylor, Russel M. (1997), *Controlling Force Feedback Over A Netwok*, in Proceedings of the Second PHANToM Users Group Workshop, Dedham, MA, USA

Sensable Technologies (2001a), *GHOST Programmers Guide*

Sensable Technologies (2001b), *PHANToM product information*, www.sensable.com

Shneiderman, Ben (1998), *Designing the User Interface*, 3 edn., Addison-Wesley

Shneiderman, Ben and Maes, Patty (1997), *Direct manipulation vs. software agents: A debate*, ACM interactions, vol. 4(6), pp. 42–61

Siio, Itiro and Mima, Yoshiaki (1999), *IconStickers: Converting computer Icons into ral Paper Icons*, in Proceedings of Human-Computer Interaction International, vol. 1, Munich, DE, pp. 271–275

Silberschatz, Abraham and Galvin, Peter Baer (1998), *Operating System Concepts*, 5 edn., Addison Wesley Longman

Sines, P. and Das, B. (1999), *Peltier Haptic Interface (PHI) for Improved Sensation of Touch in Virtual Environments*, Virtual Reality, vol. 4(4), pp. 260–264

Srinivasan, Mandayam A. and Basdogan, Cagatay (1997), *Haptics in Virtual Environments: Taxonomy, Research Status, and Challenges*, Computers and Graphics, vol. 21(4), pp. 393–404

Staples, Dan (1999), *Haptics: From Basic Principles to Advanced Applications*, chap. Haptic Application Issues, no. 38 in Course Notes for SIGGRAPH '99, ACM

Steward, Nigel, Leach, Geoff and John, Sabu (1998), *An Improved Z-Buffer CSG Rendering Algorithm*, in 1998 Eurographics/SIGGRAPH Workshop on Graphics Hardware, Lisbon, PT, pp. 25–30

Stoakley, Richard, Conway, Matthew J. and Pausch, Randy (1995), *Virtual Reality on a WIM: Interactive Worlds in Miniature*, in Proceedings of CHI 1995, Denver, CO, USA, pp. 265–272

Summers, Ian R., Chanter, Craig M., Southall, Anna L. and Brady, Alan C. (2001), *Results from a Tactile Array on the Fingertip*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 26–28

Taylor, Russel M., Robinett, Warren, l. Chi, Vernon, Brooks, Fredrik P., Wright, William V. et al. (1993), *The Nanomanipulator: A Virtual-Reality Interface for a Scanning Tunneling Microscope*, in Proceedings of the SIGGRAPH 1993 annual conference on Computer Graphics, Anaheim, CA, USA, pp. 127–134

Taylor, Russell M., Hudson, Thomas C., Seeger, Adam, Weber, Hans, Juliano, Jeffrey et al. (2001), *VRPN: A Device-Independent, Network-Transparent VR Peripheral System*, in Proceedings of the ACM Symposium on Virtual Reality Software & Technology 2001, Banff, CA

Tognazzini, Bruce (1993), *Principles, Techniques, and Ethics of Stage Magic and Their Application to Human Computer Interface Design*, in Proceedings of INTERCHI '93, Amsterdam, NL, pp. 355–366

Ullmer, Brygg and Ishii, Hiroshi (1999), *mediaBlocks: Tangible Interfaces for Online Media*, in Proceedings of CHI 1999, vol. Extended Abstracts, Pittsburgh, PA, USA, pp. 31–32

Ullmer, Brygg, Ishii, Hiroshi and Glass, Dylan (1998), *mediaBlocks: Physical Containers, Transports, and Controls for Online Media*, in Proceedings of the SIGGRAPH 1998 annual conference on Computer Graphics, Orlando, FL, USA, pp. 379–386

van Erp, Jan B.F. and van Veen, Hendrik A.H.C. (2001), *Vibro-Tactile Information Presentation in Automobiles*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 99–104

Van Scoy, Franses L., Kawai, Takamitsu, Fullmer, Angela, Stamper, KEvin, Wojciechowaska, Iwona et al. (2001), *The Sound and Touch of Mathematics: a Prototype System*, in Proceedings of the sixth PHANToM Users Group Workshop, Aspen, CO, USA

Voorhorst, F.A. and Krueger, H. (1999), *User-friendly by making the interface graspable*, in Proceedings of Human-Computer Interaction International, vol. 1, Munich, DE, pp. 416–420

Wanger, Len (1998), *Haptically Enhanced Molecular Modeling: A Case Study*, in Proceedings of the Third PHANToM Users Group Workshop, Dedham, MA, USA

Watt, Alan and Policarpo, Fabio (1997), *The Computer Image*, Addison-Wesley

Wiegand, T.F. (1996), *Interactive Rendering of CSG Models*, Computer Graphics Forum, vol. 15(4), pp. 249–261

Woo, Mason, Neider, Jackie, Davis, Tom and Shreiner, Dave (1999), *OpenGL Programming Guide*, 3 edn., Addison-Wesley

Yankelovich, Nicole, Levow, Gina-Anne and Marx, Matt (1995), *Designing SpeechActs: Issues in Speech User Interfaces*, in Proceedings of CHI 1995, Denver, CO, USA, pp. 369–376

Young, Peter, Chen, Tom, Anderson, David, Yu, Jiang and Nagata, Shojiro (1997), *LEGOLAND: Multi-Sensory Environment for Virtual Prototyping*, in Proceedings of the Second PHANToM Users Group Workshop, Dedham, MA, USA

Yu, Wai, Guffie, Kenneth and Brewster, Stephen (2001), *Image to Haptic Data Conversion: A First step to Improving Blind People's Accessibility to Printed Graphs*, in Proceedings of Eurohaptics 2001, Birmingham, UK, pp. 87–89

Yu, Wai, Ramboll, Ramesh and Brewster, Stephen (2000), *Haptic Graphs for Blind Computer Users*, in Proceedings of the Haptic Human-Computer Interaction Workshop, Glasgow, UK

Zhai, Shumin, Buxton, William and Milgram, P. (1996), *The Partial Occlusion Effect: Utilizing Semitransparency in 3D Human-Computer Interaction*, ACM Transactions on Human-Computer Interaction, vol. 3(3), pp. 254–284

Zilles, C. B. and Salisbury, J. K. (1995), *A Constraint-based God-object Method For Haptic Display*, in Proceedings of IEE/RSJ International Conference on Intelligent Robots and Systems, pp. 146–151

Zorcolo, Antonio, Gobetti, Enrico, Pili, Piero and Tuveri, Massimilliano (1999), *Catheter Insertion Simulation with Combined Visual and Haptic Feedback*, in Proceedings of the first PHANToM Users Reserach Symposium 2000, Heidelberg, DE

List of Figures

List of Tables