DOCTORAATSPROEFSCHRIFT

2008 | School voor Informatietechnologie
Kennistechnologie, Informatica, Wiskunde, ICT

Dynamic Interactions for Networked Virtual Environments

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

Pieter JORISSEN

Promotor: prof. dr. Wim Lamotte

Universiteit Maastricht

universiteit hasselt

DOCTORAATSPROEFSCHRIFT

2008 | School voor Levenswetenschappen

Dynamic Interactions for Networked Virtual Environments

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

Pieter JORISSEN

Promotor: prof. dr. Wim Lamotte

Universiteit Maastricht

universiteit hasselt

D/2008/2451/1

Acknowledgments

A researcher's work is influenced by many things: the work of others in the field, his colleagues, discussions at the bar with friends, family and ordinary daily events. So, my first reaction would be to thank everyone that has ever influenced me. Acknowledging all of them personally is a nearly impossible task and would lead to an almost endless list of names. Therefore, I will try to focus here on those that have contributed to my work directly in some way or another.

First and foremost, I would like to thank prof. dr. *Wim Lamotte* for his supervision and expert guidance. Without his support, this work would never have been what it is today. Also, I want to thank him and all other past and present members of the NVE research group: *Stijn Agten, Bart Cornelissen, Jeroen Dierckx, Bjorn Geuns, Tom Jehaes, Jori Liesenborgs, Patrick Monsieurs, Peter Quax, Miguel Rujula* and *Maarten Wijnants* for the inspiring years both on professional and personal level. Some other colleagues from other departments were also very helpful, sometimes directly related to my works, sometimes not so directly. I want to thank dr. *Joan De Boeck*, prof. dr. *Fabian Di Fiore*, dr. *Jan Fransens*, dr. *Erik Hubo*, dr. *Tom Mertens*, *Lode Vanacken, Gert Vansichem*, dr. *William Van Haevre* and dr. *Tom Van Laerhoven.*

A special thanks also goes to prof. dr. *Frank Van Reeth* and prof. dr. *Eddy Flerackers*, managing director of the EDM, for believing in me and giving me the opportunity to realize this work. Furthermore, I want to thank the rest of the senior research staff at EDM prof. dr. *Philippe Bekaert*, prof. dr. *Karin Coninx*, prof. dr. *Kris Luyten* and prof. dr. *Chris Raymaekers* for their influences on me directly and through my colleagues. I would also like to express my gratitude toward *Peter Vandoren*, *Ingrid Konings* and *Roger Claes* for taking care of the formalities associated with the successful completion of my PhD.

On a more personal note, I would like to thank my family and friends for their support; especially my parents *Piet* and *Elly* for giving me every opportunity and supporting my decisions. And finally, I want to thank *Elise Willems* for all her support, for being there for me and for brightening up my life. Thank you!

Diepenbeek, January 2008.

Abstract

The field of networked virtual environments has been an active area of research in the past decades. The increasing power of contemporary computers and the lowering of hardware and connectivity costs permits people to have this technology available in their homes and workplaces. As a result, several applications and prototypes are successfully being used in several fields. Most of the research and developments have, however, been technology driven and as a result one of the main components, interaction, is much less explored.

In this dissertation we focus on realizing more realistic virtual experiences by narrowing the gap between real-life and virtual world interaction possibilities. In contrast to ad hoc approaches, we also seek to provide a more general and reusable solution. The interactive object approach provides such a solution by employing a general purpose interaction mechanism for every kind of interaction in the virtual world. This mechanism relies heavily on a feature modeling approach that allows objects to describe their own interaction possibilities. As a result, the interactions become application independent and objects unknown to the application can be inserted at run time.

The problem of realistic interactions is, however, not limited to representation and execution of interactions. It requires the consideration of several related fields, such as realistic simulation and animation, user embodiment and human computer interaction as well. Therefore, in a second phase, we developed and integrated the ExtReAM library into the interaction framework. This platform-independent library allows us to improve realism by enabling physical simulation and more dynamic animations for objects and user embodiments. Furthermore, it allows objects to contain physical properties and actions as well. By combining realistic simulation with our dynamic interaction mechanism, much more lively virtual worlds can be achieved with minimal effort, which can result in better virtual experiences and higher levels of presence if used properly.

As more natural interaction techniques can increase the user's feeling of being immersed in the virtual world, we analyze how our dynamic interaction and animation system can be utilized to create more realistic user embodiment control and interaction techniques. Therefore, we propose two new 3D interaction techniques, utilizing our framework. The first technique allows the user to interact directly in the virtual world by controlling his virtual hand with a 3D input device. We elaborate on how this was established and how this technique can be distributed among the participants with as little bandwidth consumption as possible. The second technique involves a travel technique that provides the user with haptic feedback on what is happening to his virtual counterpart. This was realized by converting rigid body simulation information into force feedback. By means of a formal usability study, we show how this haptic travel method results in an increased feeling of immersion.

Contents

List of Acronyms

AI	Artificial Intelligent
AOI	Area Of Interest
CCD	Cyclic Coordinate Descent
CS	Client-Server
CSCW	Computer Supported Cooperative Work
CVE	Collaborative Virtual Environment
DIS	Distributed Interactive Simulation
DTD	Document Type Definition
DOF	Degrees Of Freedom
EX	Experienced
HCI	Human Computer Interaction
HMD	Head Mounted Display
HT	Haptic Travel Technique
IK	Inverse Kinematics
IO	Input/Output
IT	Interaction Technique
IX	Inexperienced

JT	Jacobian Transpose
KT	Keyboard Travel Technique
LAN	Local Area Network
MMORPG	Massively Multiplayer Online Role Playing Game
MUD	Multi-User Dungeon
NVE	Networked Virtual Environment
P2P	Peer-to-Peer
PE	Participant Experience
PT	Phantom Travel Technique
PC	Personal Computer
SDK	Software Development Kit
TT	Travel Technique
VE	Virtual Environment
VR	Virtual Reality
WAN	Wide Area Network

List of Figures

General Introduction

The idea of creating a computer-generated experience that is indistinguishable from reality has since the beginning of the computer age been subject of many books, movies and theories. Would it not be amazing, if instead of going to a museum to look at some artifacts and read information charts, we could virtually travel through time and space and visit any era or location we can imagine? What if we could be completely present, able to see, hear, smell, taste and touch everything. Being able to interact with everything and everyone in a way that feels completely natural. Taking this idea a little further would lead us to the situation where we would be unable to tell the difference between our real life and a life in a simulated environment. Although these 'Matrix' or 'eXistenZ' theories are still only the subject of science fiction novels and movies, it is clear that over the past decades the Virtual Reality (VR) community has become an important part of computer science and its applications have grown into much more than games and laboratory toys.

However, even though a lot of research has been done by the VR community, we are still a long way from achieving 'virtual trips' into truly realistic virtual worlds. In order to create such overall experiences, VR applications will need to stimulate the participant's senses. Ideally, it would provide information to all the senses (as there are sight, hearing, smell, taste and touch). It is well known that over the last decades, most of the efforts trying to generate virtual worlds have been spent on the visual sense. Especially the gaming and movie industry in combination with the tremendous evolution in hardware have boosted computer graphics and animation research, resulting in excellent real-time graphical rendering of virtual worlds with animated creatures and physically simulated objects. Auditory feedback has also been investigated thoroughly, and many contemporary applications support realistic sound effects and 3D localized sound. On the other hand, olfactory and taste feedback have gained very little attention from the VR research community and although some experimental systems exist [Davide 01, Chen 06], these domains are considered to be in their early research phase. As processing power increased over the last years, haptic feedback, stimulating the sense of touch, has gained more and more interest and has grown into a mature domain. As a result, it has been successfully applied in several practical applications in the domains of robotics, telerobotics, CAD, sculpting, medical simulations and training [Stone 00], and several kinds of haptic Input/Output (IO) devices have been developed [Berkley 03, Fisch 03].

Not only computer interface technology has advanced. The advances in communication technologies have caused geographical boundaries for social interaction to be practically dissolved. The use of asynchronous information exchange systems such as the WWW and email have become omnipresent and fulfill a key role in the current workspace. The expansion of high-speed broadband Internet access, has led to the rising of synchronous communication systems such as applications for real-time communication, videoconferencing and distributed forms of VR, often referred to as Networked Virtual Environments (NVEs) or Collaborative Virtual Environments (CVEs). However, the deployment of these systems has been much less preeminent.

Thus, the technology to enable realistic NVEs has advanced enormously, and the research done resulted in several standards allowing the creation and distribution of different kinds of data necessary to support such environments. Furthermore, several VR applications are successfully being used in several fields such as surgery training, flight simulators, networked shared environments for teleconferencing, human factors analysis, training, education, virtual prototyping, simulation-based design and entertainment and many more. The increasing power of contemporary computers combined with the lowering of hardware and connectivity costs permits people to have all this technology available in their homes and workplaces. So why is it that we are still working on 2D desktops? Why are we still sending emails typed on a keyboard? And how come the only widely known applications of VR are the commercial computer games? Why don't we make use of all the technology that is available at our fingertips?

Problem Statement

One way of explaining the slow adoption of 3D VEs may be the lack of natural ways of interacting with these systems. Most of the research and developments have been technology driven, focusing on better graphics, larger numbers of simultaneous users, more extensive and detailed environments, more realistic feedback, supporting more devices, etc. Yet one of the main components of VEs, *interaction*, is much less explored. Especially collaborative systems and video-based distributed systems have tended to focus on simple interactional situations and have mostly avoided the use of more advanced interactions such as those involving advanced animation techniques and realistic simulation of physical objects. It seems as if the quest for larger, more realistically-looking VEs, with massive multi-user support and streaming multimedia contents has overshadowed the research on how we could perform even the simplest of tasks, such as picking up an object or pointing at a location in the virtual world. Although interaction and interactivity have been studied, a gap still remains between real world richness of possible actions and their virtual counterparts. As an example, consider the seemingly trivial action of opening a door. In the real world it is so common that nobody even needs to think of how to do it; however, when we try to come up with a general solution for NVEs, it's a whole different story. It requires taking into account aspects of user embodiment, animation, collision detection, object manipulation, etc. Furthermore, most NVE developers consider interaction to be a by-product of the VE design, a necessary aspect to demonstrate other components. As a result, most interaction approaches have been developed in an ad hoc fashion. Consequently, it is very hard to find relevant information on how exactly VE interactions are implemented and which underlying mechanisms are used in these systems, since most publications hardly spend any attention on the matter. As a result, although many systems are graphically very realistic, with respect to interaction, most professional VE systems are fairly static and focus on communicational aspects. Most systems only allow the most basic tasks, such as travel through the virtual world. Most other professional interactive systems have focused on 3D modeling aspects, allowing users to select and move static objects in the environment.

Contributions

The overall purpose of this work is to investigate how more lively and realistic VE experiences can be achieved. Instead of focusing on the graphical aspects, we try to achieve this by narrowing the gap between real-life and virtual world interaction possibilities. In contrast to ad hoc approaches, we seek a more general solution that is able to support NVE requirements. Therefore, a new, general way of modeling and executing VE interactions is required. The problem of realistic interactions is however not limited to their representation and execution mechanism. It requires the consideration of several related fields, such as realistic simulation and animation, user embodiment and human computer interaction as well. Moreover, to enable more advanced useful interactions, all these interaction components need to be attuned to each other. How this can be realized is also the subject of this investigation.

This dissertation reports on our efforts in creating more interactive VE solutions and describes:

- the design, implementation and evaluation of a new general way of providing dynamic interaction for VEs: the *interactive object approach.* This consists of a new general-purpose interaction mechanism enabling more dynamic VEs. The proposed system is designed to deal with all possible interactions in a virtual world. The idea is to construct a world using only *interactive objects* that contain their own interaction information. As a result, the object interactions become application-independent and only a single interaction scheme is required to handle all virtual interactions in the VE. Also, the system allows for instant integration of new, ‘unknown’ interactive objects during simulation;

- the *ExtReAM* library, a plug-in-based animation library. This library will enable us to improve realism by enabling physical simulation and more dynamic animations for objects and user embodiments which can result in better VE experiences and higher levels of presence if used properly. *ExtReAM* is built around an object-oriented, platform-independent core that can easily be extended with plug-ins. While the core system provides functionality for managing plug-ins and objects, the plug-ins are responsible for more specific tasks such as object loading and various animation techniques. Different plug-ins can be used for different platforms, when necessary, and plug-ins need to be loaded only when their functionality is required. In this way, *ExtReAM* is prepared for next generation VEs on different platforms;

- the *physical interactive object approach*, an extension of the basic interactive object approach, that increases the realism of the interactive world by employing rigid body simulation to calculate all actor and object movements. Furthermore, it allows objects to contain physical actions and properties as well. The extended animation and simulation functionalities are provided by integrating the *ExtReAM* library. By combining realistic simulation with our dynamic interaction mechanism, much more lively virtual worlds can be achieved with minimal effort;

- new techniques for interacting with virtual environments. We show how inverse kinematics can be used to increase the interaction possibilities through realistic direct avatar control. This allows for easy, on-the-fly creation of new interactions. Furthermore, we present a way to couple stable haptic force feedback to rigid body simulation in order to achieve haptic travel in dynamic VEs. Through the use of a haptic IO device, we provide the user with a realistic way to control an animated avatar’s travel in a VE. Furthermore, we show how several physically simulated forces based on changes in the terrain and avatar collisions, influence travel and give the user feedback on what is happening to its virtual representation; increasing his feeling of presence;

- formal and informal evaluations of all of the proposed solutions and techniques through experimentation, integration in real-world applications or usability testing.

Although the field we are working in is closely related to the rising field of virtual autonomous actors, we must stress that this is not the field we are considering. However, there are several links, and where they appear, they will be mentioned. Another field that is closely related to NVEs is the field of computer games. The main difference between games and the NVEs we consider is that in contrast to games, that aim at entertaining the user and trying to have him play as much as possible, professional NVEs are mostly created with a specific goal in mind. They are tools for achieving a mutual goal. These goals can include simulation, education, training, a medical procedure, etc. As a result of their difference in nature, games tend to outperform professional NVEs with respect to usability and certainly aesthetics. Professional NVEs tend to spend less attention on these subjects, and NVE developers focus more on correctness and quality of the tasks at hand. It is clear that both fields can learn from each other, therefore, we will investigate techniques that are often only used in games, but that can be useful for professional NVEs, as well.

Outline

This dissertation comprises five parts:

1. Part I gives a general overview of the research field and discusses some important definitions and related work.
2. Part II shows how a feature modeling approach is used to enable dynamic interactions in virtual environments.
3. Part III discusses the importance of realistic simulation and animation and presents how we constructed a flexible animation library to extend our dynamic interaction approach.
4. Part IV proposes two dynamic interaction techniques employing realistic animation, enabling more realistic avatar control.
5. Part V presents some closing remarks and directions for further research.

In Part I we start by defining some of the terminology that is used throughout this dissertation and give some background information on the field of VR systems in Chapter 1. This overview is completed in Chapter 2 by a historical overview of NVE systems and how user embodiments have evolved in VR applications. Readers who are familiar with the general concepts of (Networked) VEs, interaction, data distribution and user representation can skip this part.

Part II focuses on how more dynamic interactive NVEs can be supported. Chapter 3 introduces the problem with contemporary interaction approaches and their lack of generality. An overview of related work concerning interaction modeling is provided in Chapter 4. Next, Chapter 5 presents the first version of the interactive object approach for NVEs, which is based on object generalization and utilizes a feature modeling approach. It is designed to support run-time adjustable interactions between all interactive objects in the virtual world by a single interaction mechanism. How interactive objects are simulated and controlled are also discussed. Finally, Chapter 6 concludes this Part with some general remarks and gives pointers to improve realism.

The third Part elaborates on how our interactive object platform can be extended to support more realistic interactions. Chapter 7 explains how dynamic animations and simulations can be used to provide more realism for interactions. The related work is discussed in Chapter 8. In Chapter 9 we present *ExtReAM*, a new animation and simulation library that can easily be extended with new animation techniques and is easy to integrate. Chapter 10 then discusses how *ExtReAM* is integrated in the interactive object approach, enabling new animation and simulation functionality and resulting in more realistic dynamic interactions. Our findings are given in Chapter 11.

Part IV discusses the HCI part of the interaction process. Chapter 12 elaborates on the link between interaction techniques, interaction and embodiment animation. Also it discusses how more natural interaction techniques can improve the acceptance of VR for a larger audience and how it increases the user's feeling of being immersed in the virtual world. Chapter 13 presents related work in the field of 3D interaction techniques and defines the concepts of presence and workload and describes some of the most important methodologies for measuring these. In Chapter 14 we discuss how dynamic inverse kinematic animation can support more realistic avatar interactions in virtual worlds. As we discussed earlier, dynamic animations are often avoided in NVEs since they require more networking resources than predefined animations. Therefore, we will also elaborate on how this new interaction technique can be distributed as efficiently as possible. Thereafter, in Chapter 15 we discuss how rigid-body simulation can be exploited to generate haptic feedback forces. Furthermore, we present a haptic travel method that allows users to navigate through a virtual world while receiving haptic feedback on what happens to their virtual counterpart. After presenting the technique, we evaluate the approach in a formal experiment that analyzes the influence of haptics on the user's feeling of immersion and workload during travel in a virtual world in Chapter 16. Conclusions for this component of our research are given in Chapter 17.

Part V discusses the overall conclusions of this dissertation. Furthermore, pointers for further research are given.

Part I

BACKGROUND

Table of Contents

CHAPTER 1

Defining Interactive Virtual Reality Applications

In this chapter we define the concepts that are used throughout this dissertation and describe the areas we are working in.

We first take a look at VR in general, both on the level of the software system as well as the ways one can interact with such an environment. We define the different components of interest and their properties. Furthermore, we describe how VR and virtual simulation can be shared among different clients over a network. In the following chapter, we complete this overview by giving the history of NVE systems and user embodiments. In case the reader is already familiar with the concepts of VR, VEs, interaction and data distribution, these two chapters may be skipped.

1.1 VR, VE and NVE

The term Virtual Reality has been utilized in several fields and is hard to define. Perhaps the reason for this difficulty is exactly caused by the widespread use of the term. Belleman [Belleman 03] states that the term is used by among others: game developers, arts movements, lyricists, visionaries, ... Also, several synonyms have become common, e.g. artificial reality, simulated reality, cyberspace. We will use the term VR to refer to all computer-based systems that support interactive virtual worlds, or Virtual Environments (VEs).

Throughout the history of VR research, several definitions of the term VE have been proposed. Perhaps the most adopted is the one given by Witmer et al. [Witmer 96]:

> "A Virtual Environment is a computer-generated simulated space with which an individual interacts."

This definition was later expanded by Singhal en Zyda [Singhal 99] in order to define Networked Virtual environments. This led to the following definition:

> "A Networked Virtual Environment is a software system in which multiple users interact with each other in real-time, even though those users may be located around the world."

In order to distinguish NVEs from other kinds of systems, the authors also describe five essential features for classifying systems as an NVE. They need to support:

1. a shared sense of space,
2. a shared sense of presence,
3. a shared sense of time,
4. a way to communicate,
5. a way to share.

From these definitions we can easily deduct that VEs and NVEs are multidisciplinary software systems, trying to provide the users with a sense of realism and (shared) experience wherein they can interact. In order to achieve this goal, (N)VEs first of all require a simulated space or world which has to be represented to the users, involving computer graphics as a first important terrain of expertise. Secondly, users must be able to interact with the environment, creating a link with the field of HCI. For NVEs, another important part is the network component, which has to ensure that the simulated environment is synchronized among its users and all interactions are being distributed to all participants, linking us into the field of computer networks. So as we can see, in order to create useful VEs and NVEs we need to apply techniques from all of these areas and efficiently combine those into a single system. It is exactly the combining of these different fields, that each on their own have an impact on the end system and on the other parts that makes NVE systems one of the most challenging research areas in computer science.

In this work, the emphasis lies on the interactions occurring in NVEs, which is a crucial part that has often been overlooked, which is striking, as it is an important part of the definitions of both VEs and NVEs.

1.2 Interaction, Interaction Techniques and Input Devices

Interaction is a subject that has been widely studied in many areas and, as a result, has different tailored meanings in various sciences. In sociology, for example, interaction is defined as a dynamic, changing sequence of social actions between individuals (or groups) who modify their actions and reactions due to the actions by their interaction partner(s)[1]. In physics, an interaction or force specifically refers to the action of one physical object upon another resulting in potential energy - the physical objects under consideration may range from point particles to quantum fields. A more general definition of interaction states that interaction is a kind of action that occurs as two or more objects have an effect upon one another[1]. [APA 07] on their part define the term quite similarly as: a reciprocal action, effect, or influence.

The focus of this study is in the field of NVEs, therefore we adapt the latter definition for our purposes as follows:

> "A virtual interaction is an event occurring in a virtual environment that has an impact on one or more objects in the virtual world or on the world itself."

By object, we mean any entity that is present in that virtual environment, thus also including representations of human or AI controlled actors. We deliberately leave out the specification of how and by whom this event is triggered in order to remain as general as possible. In the remainder of this work, if we discuss interaction, we mean these virtual interactions. In relation to interaction, we must make a clear distinction between what we refer to as *interactions* and *interaction techniques*, two terms that are too often used incorrectly. Interaction techniques (ITs), also named *interaction methods* or *methodologies*, are used to refer to the way that user input is mapped onto

[1]Wikipedia - Interaction (http://en.wikipedia.org/wiki/Interaction)

executed actions in the VE. ITs are used to trigger interactions in the VE, for example by mapping user gestures captured by an input device into actions that are performed by the user's virtual representation. An illustration of how users, input devices, ITs and interaction relate to one another is given in Figure 1.1. For clarity, the interaction mechanism itself is ignored.

Input devices provide the user with an interface to the virtual world through the IT. The input devices are used to register the user's gestures and provide them to the IT. In order to be efficient, ITs should be intuitive and accurate. They should fulfill the user's intentions fast and correctly. Interaction and ITs play an important role in the creation of the feeling of *presence*, the sense of *being there*. Good ITs and realistic methodologies can greatly improve the experience. On the other hand, methods that force the user to step out of the VE experience and into the real world, even for a brief moment, can totally break the feeling of presence and ruin the experience. As a result, over the past decades, several sorts of input have been proposed, in all kinds of shapes, with varying Degrees Of Freedom (DOF) [Berkley 03, Fisch 03]. It is the DOF that define the expressiveness of an input device. Often, in order to achieve a higher expressiveness, several input devices are combined for example by using two-handed input [Hinckley 98, Casalta 99]. The devices and techniques used in a VE are also very dependent on the type of VR system and the tasks that need to be performed. Immersive VE systems, employing an immersive display such as a HMD or a CAVE, will usually use less tactile approaches such as tracking sensors, while desktop VEs will often use more grounded devices.

West and Hubbold [West 98] have argued that although the hardware makes it possible to display visually rich environments, the ways in which users can interact in those environments remain sadly inadequate. Furthermore, a major part of the problem has to do with improvements in software support for modeling environments in order to support richer forms of interaction. In practice, this resulted in the fact that very few toolkits for VE application developers are available that facilitate the construction of interactive VEs. The ones that do exist, focus on, or are specifically tailored for specific (groups of) input hardware, offer limited flexibility, or are specific to the application area for which they were developed. This is in contrast to the many ITs that have been proposed [Mine 95, Hand 97, Bowman 99, Subramanian 00], and the fact that many of them have been applied effectively in VE applications. Unfortunately, investigations show that the interaction possibilities within most NVEs often remain fairly limited (navigate, select and move object), resulting in seemingly static scenes with only a limited interactivity.

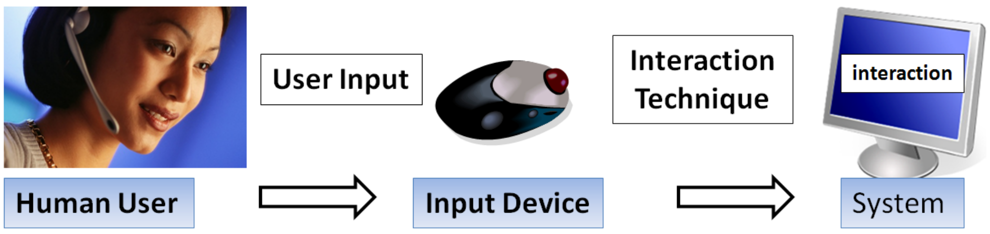


Figure 1.1: The different steps and components of how a user interacts in a VE

1.3 Interaction Classes

ITs are often classified from an application's perspective. A classic example is the classification of Bowman et al. [Bowman 99] which categorizes ITs into four groups: application control, viewpoint motion control, object selection and object manipulation. Interactions, on the other hand can be best categorized by the types of objects that interact with each other, and more specifically by the entity that triggers the interaction. Herein, *actors* are often distinguished from other objects. We see actors as entities that are in some way in control of their actions. This includes both human controlled avatars as well as autonomous agents. As a result, following categories of interactions are distinguishable:

1. Actor → Object,
2. Object → Object,
3. Object → Actor,
4. Actor → Actor.

The interactions themselves are usually classified as collision, transform actions (move, rotate or scale) or a more complex interaction. Furthermore, the combining of several interactions results in even more possible combinations (e.g. Actor → Object → Actor, etc.).

Most single-user simulated systems are limited to the first two categories of interactions. As an example of category one, consider the case of an actor (human user or an autonomous agent) grasping an object and moving it to a new position. For the second group of interactions we can consider the case of a knocked over domino, toppling another domino. *Object - Actor* interactions are much less employed in VR systems. Haptic systems provide perhaps the best example, where a moving object can push the user's representation away from its location. Finally, the last category of interactions are by definition only possible in NVEs. They form the biggest challenge for developers and IT designers, and form an issue that has not yet been solved (except for some specific applications and setups). As Glencross et al. point out, there are two main problems related with this kind of interaction: the first concerns the choice of simulation methods, and the second is synchronization of state information across participants involved in the collaboration [Glencross 05]. Some solutions specifically tailored for distributed haptic VEs can also be found in [Glencross 05]. In the next section we will discuss how simulated VEs can be distributed and synchronized among different participants in NVEs.

1.4 State Synchronization and Distributing Simulation

Although it is not the main topic of this dissertation, as we are working in the field of NVEs, we must take into account that all users need to have a consistent view of the VE. In fact this is one of the most important aspects of NVEs. It would be practically impossible for two or more participants to collaborate if they did not have a consistent view of the shared virtual world at all times. When we consider synchronizing interactive, simulated NVEs, two important issues need to be considered. The first concerns the architecture of how state updates are distributed between different participants. The second has to do with deciding where the simulation is run and which nodes of the distribution architecture are responsible for solving conflicting interactions. Exchanging state information between participants on small scaled Local Area Networks (LANs) is generally fast and reliable, however a Wide Area Network (WAN) such as the Internet suffers from network delays and jitter, making it very hard for NVE developers to create perfectly synchronized views for all participants.

It is clear that the choice of distribution architecture has an important impact on the choice of where the simulation is performed and vice versa. Furthermore, these choices are largely dependent on the underlying network, the application and the types of data that needs to be distributed. Most high-end systems use hybrid approaches where different data kinds with different distribution needs are distributed in different ways. As an example consider a video frame from a playing video stream versus a text message from a discussion. Video frames are normally sent unreliably, as minimal delays are important, and a re-sent lost video frame will arrive too late anyway. On the other hand, it is important that all other participants receive a chat message in order to have the same information. The timing is however less strict, thus a reliable technique is therefore mostly applied. Next we will discuss distribution architectures and solutions for distributed simulation.

1.4.1 Distribution Architectures

In order to be able to give all users a consistent view of a VE and facing the issues of networks, NVE systems usually adopt one of the two basic distribution architectures: Client-Server (CS) or Peer-to-Peer (P2P).

The simplest, and probably most applied architecture is the CS solution (see Figure 1.2(a)). This employs a single, usually dedicated network node (the server), that is responsible for the distribution of updates to all users (the clients). In this way, the users do not need to know about the other users, they just send updates or update requests to the server, which has knowledge of, or connections to all users and forwards the information to those that require the update. This approach has several advantages. First of all, clients only need to know the address of the server, and can work independently of other clients joining or leaving the NVE session, simplifying the network component of the clients. Also, implementing this approach and upgrading the server is relatively simple and requires minimal effort. On the downside, using the CS architecture creates a single point of failure. If e.g. a server crash occurs, the NVE is unavailable for all users. Also, the server must be able to handle all users, and their network traffic, creating a bottleneck situation. For some applications, which allow a large number of users, this can be problematic. Several solutions for these issues exist, such as backup servers or splitting up servers, or splitting up the environment in different areas, managed by different servers, without the user's knowledge. These servers would, however, need to be synchronized very strictly, resulting in other difficulties. Another downside of the CS approach is that since all network messages need to travel via a server to the other clients, extra delays are introduced. On the positive side, if the server is designed more intelligently than just forwarding all incoming updates to all other users, it can significantly reduce network traffic.

As an alternative to CS, the P2P architecture does not employ a central server for data distribution (Figure 1.2(b)). Each client distributes its updates to the other clients directly. This approach certainly solves most of the problems with CS, such as the single point of failure, the network bottleneck and the extra delays. On the other hand, it introduces many other issues. All users need to maintain a list of other clients and need to handle users that join and leave the NVE. As a result the clients need to perform more operations and will also use much more network traffic. Furthermore, since clients need to handle all the consistency rules themselves it is much harder to implement and client software will be much more complex. Finally, in case of extra bandwidth requirements, it is much harder to upgrade for example all client's networks. This, in contrast to the CS architecture where the server can be upgraded independently of the clients. P2P architectures appeared a few years earlier than CS systems. This probably resulted from the fact that using a dedicated server was very expensive in the beginning years of NVEs [Joslin 04].

Hybrid approaches employ techniques from both CS and P2P architectures, resolving the issues that come with the employment of one architecture. For example a single machine can be used to perform some specific management tasks but without it being used to distribute all the data as a server would do in the CS approach, resolving the network bottleneck. Another type of hybrid approach employs different architectures for different kinds of data. For example video can be streamed via a server while position updates among clients are sent in a P2P way. Many other kinds of hybrid approaches exist, however discussing all of them is beyond the scope of this work.

On top of choosing the right architecture, NVE designers must also choose between different network protocols (TCP, UDP, multicast), choose reliability parameters (reliable UDP, unreliable multicast,...) and can implement several improvements over dumb broadcasting of all updates to all users (dead reckoning techniques [Singhal 95], area of interest management policies [Boulanger 06], compression techniques,...). Since the focus of this work is on the interaction within VEs and NVEs, we will not elaborate further on the subject of network architectures and efficient data distribution. More details on this subject can be found in [Macedonia 97, Quax 07].

1.4.2 Distributing Simulation

Similarly as on the network level, at the simulation level, NVE designers need to make choices on which network entities are responsible for keeping the simulation consistent and resolve conflicting interactions. Computer Supported Cooperative Work (CSCW) locking based floor control techniques [Dommel 97] are out of the question, since in dynamic NVEs, we want to allow interaction

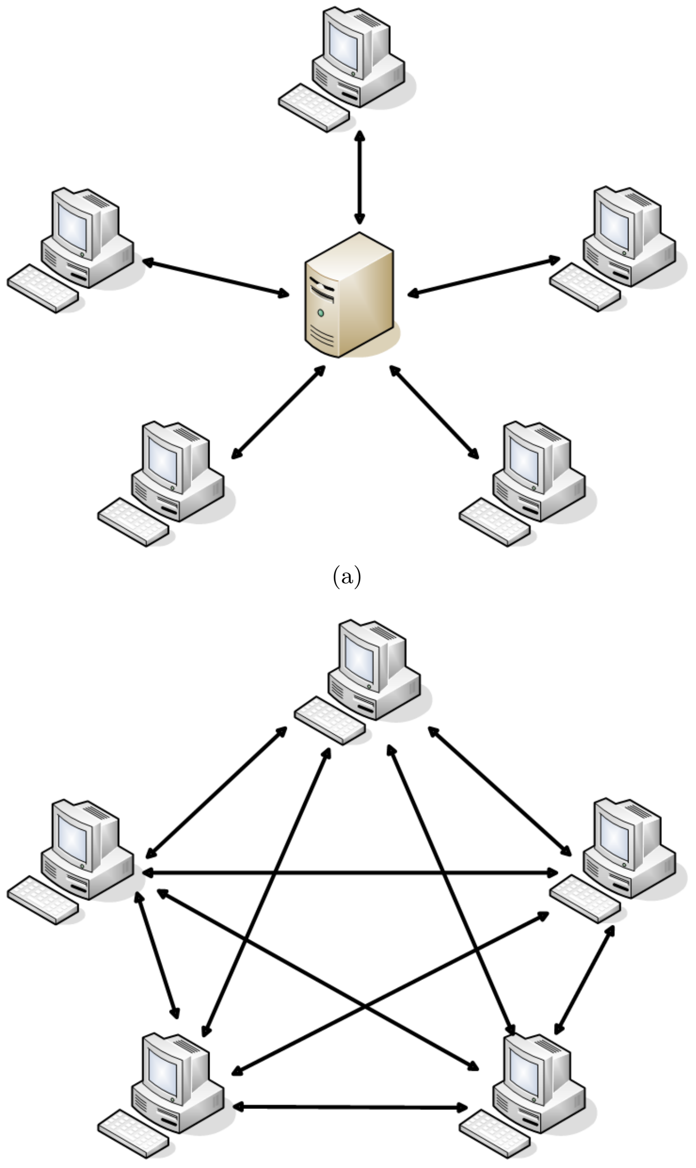


Figure 1.2: The two basic distribution architectures: (a) Client-Server and (b) Peer-to-peer.

with any object at any time even if another user might already be interacting with that object. What is more, this kind of cooperation is one of the most important aspects of NVE systems.

The traditional approach is similar to the CS architecture. It uses a *single simulation controller* that runs the simulation (usually the server when built on top of a CS architecture, but it could also be a selected user machine). The other machines just send interaction requests to this controller, which applies them to the VE and distributes the resulting changes to all entities. This approach is simple and easy to implement. Furthermore, since only one machine is responsible for all the interactions, conflicting interactions can relatively easily be detected and resolved. A high-end machine can handle many interactions, and by applying this approach, all other clients are alleviated of the simulation and conflict handling tasks, which can become very hard, when for example physical realism is required. On the downside, similar disadvantages as with the CS approach result from this single point approach. The biggest disadvantage is the round trip delay that is introduced while interacting. When the users wants to perform an interaction, the request must be send to the server, the interaction needs to be performed there and the resulting update needs to be transmitted back to the user before he can see the impact of his interaction. Obviously, this approach is therefore only applicable for situations where the delays are small, as the response times for interactive applications may not become too large in order to maintain a good experience. Studies have shown that NVE users are able to adapt to latencies of up to about 200ms [Park 99] , however, this is highly dependent of the task that is being performed. Furthermore, jitter also plays an important role in the user's ability to cope with delay [Quax 07].

As a solution to this approach, some systems use the single controller approach, but with (partial) simulation on the client's machine as well, in this way, if no conflict occurs, the users do not suffer from the round trip delays, as they perform the interaction results on their own. However they still send the interaction requests to the server, and other users are only informed of changes when they are applied by the server. In case of a conflict, the inconsistent states are corrected by the server as soon as the server sends updates. This approach obviously requires more processing from the clients and is harder to implement.

At the other hand, similar to the P2P network architecture, a totally *distributed simulation* is also possible. In this approach, no single fixed entity is responsible for the entire simulation or conflict handling. As a result, the clients need to be able to simulate the entire virtual world themselves and need to resolve conflicting interactions by negotiation or some fixed rule system. The advantages are similar to the P2P approach: less delays, no single point of failure, but it is much harder to implement, especially when one must take into account that in most NVEs users can enter and leave at any time.

While the two approaches map very well onto the two most used network architectures, they are not necessarily linked to each other. It is possible to apply a distributed simulation onto a CS distribution scheme. In this case the server could control which user machine controls which part of a simulation, and distribute messages among the clients. The other extreme of having a single controlled simulation on top of a P2P architecture is just as well possible. The clients could for example select one client as the simulation controller. As long as this user stays logged on to the NVE, he can stay in control, when he logs off, or when he is unable to control the simulation for some other reason, the simulation could be conveyed to another selected user, and so on.

In order to gain a better understanding of how NVE systems and their components have become what they are today, the next chapter gives a brief discussion of their evolution.

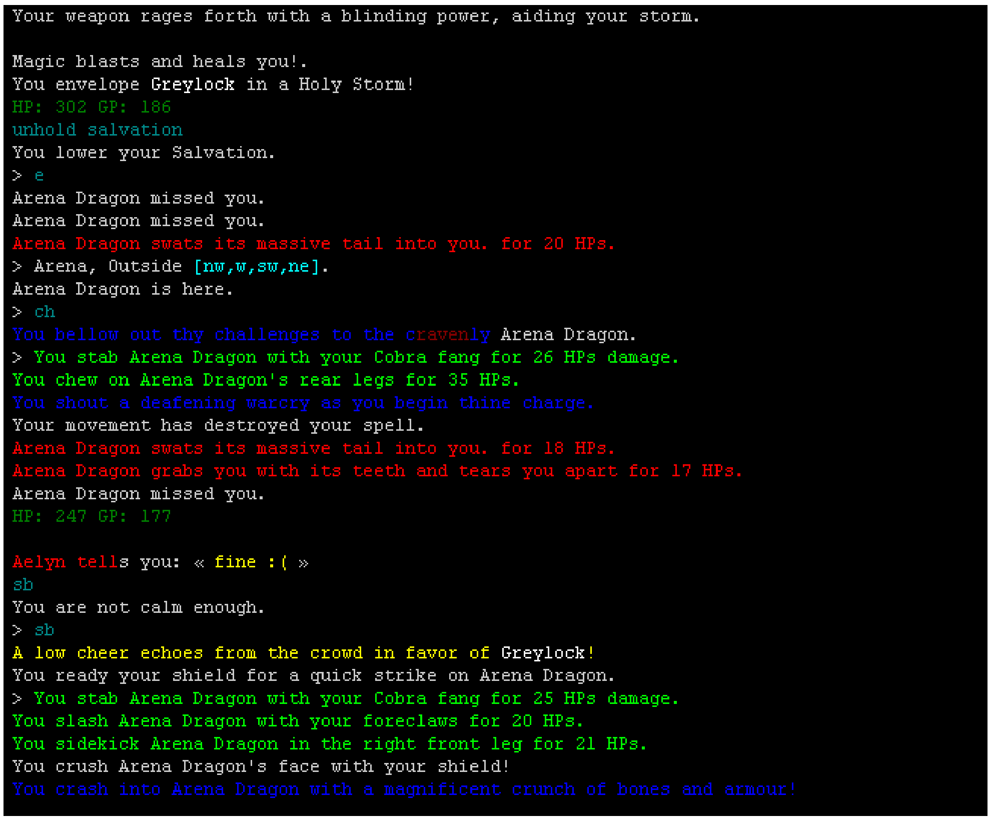
CHAPTER 2

A Brief Historical Overview

2.1 NVE systems

NVEs are the result of the merging of the fields of the VR and CSCW. CSCW and *groupware* involve all systems that use technology in order to mediate collaboration in a professional context. The roots of NVEs lie in the text-based *Multi-User Dungeons* (MUDs) that have been around since the late 1970s. The original MUDs were completely text-based multi-player role playing games. The entire environment, including objects, players and objects were described in text. Interaction with these MUDs was typically also text-based. Users could enter commands that were based on natural language (e.g. go hallway, open door, say 'hello' to Tom, look at object). The main advantage of this form of interaction is that it provided an almost infinite number of DOF with standard hardware. This is also one of its major drawbacks, as it is almost impossible to limit the DOF, unless by allowing only a limited part of the vocabulary. Figure 2.1 shows an example of such a text-based MUD.

With the uprise of Personal Computers (PCs) and computer graphics technology in the 1980s, MUDs evolved into graphical MUDs. Some systems merely used graphics to enhance the text-based visualization of the world,



```
Your weapon rages forth with a blinding power, aiding your storm.

Magic blasts and heals you!.
You envelope Greylock in a Holy Storm!
HP: 302 GP: 186
unhold salvation
You lower your Salvation.
> e
Arena Dragon missed you.
Arena Dragon missed you.
Arena Dragon swats its massive tail into you. for 20 HPs.
> Arena, Outside [nw,w,sw,ne].
Arena Dragon is here.
> ch
You bellow out thy challenges to the cravenly Arena Dragon.
> You stab Arena Dragon with your Cobra fang for 26 HPs damage.
You chew on Arena Dragon's rear legs for 35 HPs.
You shout a deafening warcry as you begin thine charge.
Your movement has destroyed your spell.
Arena Dragon swats its massive tail into you. for 18 HPs.
Arena Dragon grabs you with its teeth and tears you apart for 17 HPs.
Arena Dragon missed you.
HP: 247 GP: 177

Aelyn tells you: « fine :( »
sb
You are not calm enough.
> sb
A low cheer echoes from the crowd in favor of Greylock!
You ready your shield for a quick strike on Arena Dragon.
> You stab Arena Dragon with your Cobra fang for 25 HPs damage.
You slash Arena Dragon with your foreclaws for 20 HPs.
You sidekick Arena Dragon in the right front leg for 21 HPs.
You crush Arena Dragon's face with your shield!
You crash into Arena Dragon with a magnificent crunch of bones and armour!
```

Figure 2.1: An example of a text-based MUD interface.

while others evolved into full 3D visualizations with customized user representations. The user interaction methods also evolved. With the introduction of the mouse, text-based input was being replaced by point and click, allowing the user to select objects and commands by clicking on their graphical representations, or by dragging and dropping objects toward each other in order to combine them. Figure 2.2 shows a screen from Maniac Mansion[1] employing an interface that combines natural language with a point-and-click approach. The technologies used in these MUDs were now also being used more and more by the CSCW community in order to create shared displays of information, group decision systems, multi-user editors, ... This resulted in more interactive professional collaborative applications such as Computer-Assisted Design/Computer-Assisted Manufacturing (CAD/CAM), Computer-Assisted

[1]Wikipedia - Maniac Mansion (`http://en.wikipedia.org/wiki/Maniac_Mansion`)

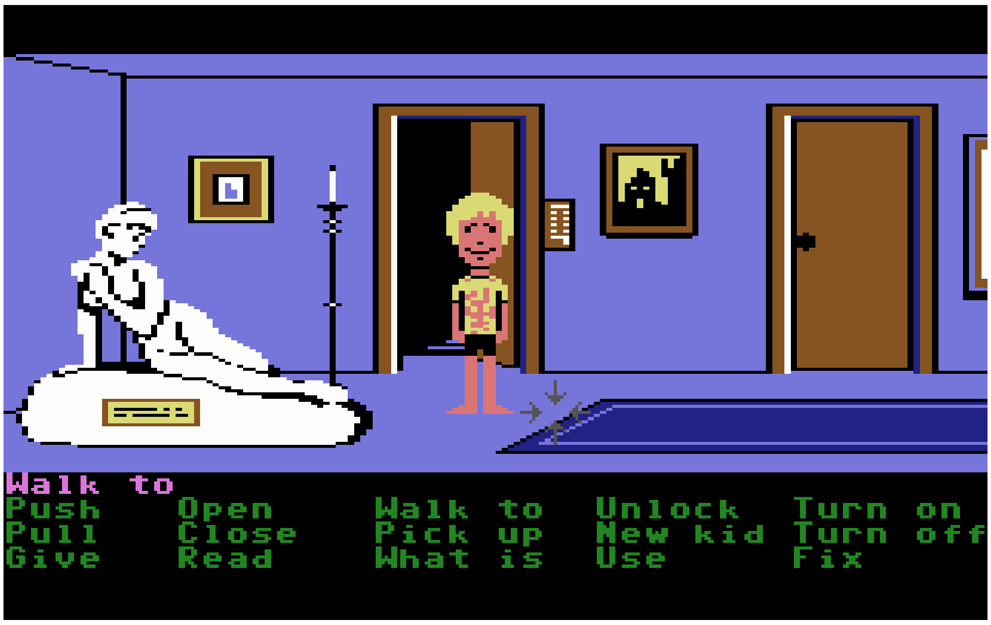
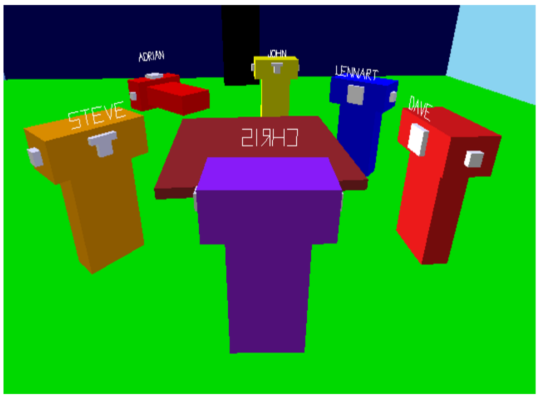


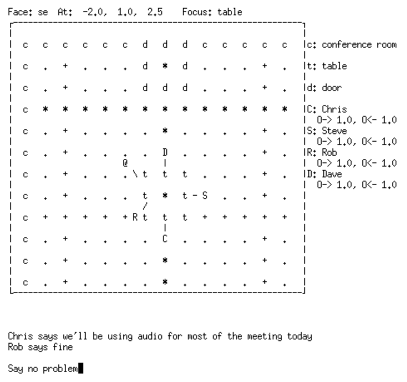
Figure 2.2: Maniac Mansion (1988). The interface is partially graphical and partially text-based, point-and-click.

Software Engineering (CASE), concurrent engineering, workflow management, distance learning, telemedicine, ...

Also in that period, the US military started to interconnect several of its single-user VR simulations in order to allow training with human allies and opponents. The SIMNET [Miller 95] project resulted in a standard that is still being used today, DIS (Distributed Interactive Simulation). A few years later, non-military NVEs started to appear. Some of the most important include: NPSNET [Macedonia 95], DIVE (Distributed Interactive Virtual Environment [Hagsand 96]) and MASSIVE (Model, Architecture and System for Spatial Interaction in Virtual Environments [Greenhalgh 95]). NPSNET, was one of the first systems that although it was based on the DIS system, was designed for the Internet, using IP multicast to inform clients of updates. Furthermore, it was one of the first systems that deployed Area Of Interest (AOI) management, in order to decrease the number of updates that needed to be sent. DIVE on the other hand, aimed at dedicated networks by employing full broadcast for all updates that needed to be sent, and all clients held a full state of the entire virtual world. It differs from similar approaches in its dynamic and flexible capabilities and its focus on interaction and human-human communication. Dynamic behaviors of objects are described by interpretative Tcl scripts evaluated on any node where the object is replicated. VEOS [Bricken 94] is a complete NVE architecture that provides integrated software to develop general applications. VEOS uses a tightly-integrated computing model for management of data, processes, and communication describing dynamic entities and their behaviors as LISP programs. MASSIVE primarily focused on teleconferencing, but grew out to be much more. It is underpinned by the so-called *spatial model of interaction* for AOI management and was one of the first systems relying on P2P unicast messages. Furthermore, the *awareness level* of objects could differ for audio and visual channels. Also, MASSIVE supported three kinds of interfaces (text, graphics and audio) that could be arbitrarily combined according to the user's equipment, a situation that is currently again a hot topic since NVEs are also evolving onto mobile platforms such as cell phones and PDAs. In MASSIVE, the users could specify their graphics embodiment using a simple geometry description and use it to show their communication possibilities. Interaction however, was limited to communication, navigation and to grasp and move virtual objects. An aura collision manager is responsible for detecting awareness collisions for each declared medium. When two auras collided, communication was made possible. An illustration of the graphical and textual interface to MASSIVE is shown in Figure 2.3. Note the resemblance between the textual interface commands and MUDs. VLNET [VLNET 07] is an exception in the field of these early NVE systems. The system has been developed at MIRALab at University of Geneva, and Computer Graphics Laboratory at Swiss Federal Institute of Technology. In contrast to other systems, VLNET, focuses on integrating artificial life techniques with virtual reality techniques in order to create truly virtual environments shared by real people, and with autonomous living virtual humans with their own behavior, which can perceive the environment and interact with participants.

With the tremendous evolution of PC hardware and performance, NVE systems boomed in the second half of the 90s, resulting in many new systems such as SPLINE (Scalable Platform for Large Interactive Networked Environments [Waters 97]), VPARK (an extension of VLNET, creating a virtual amusement park [Seo 00]), BAMBOO ([Watsen 98]), and many more. Furthermore, existing systems evolved as well, resulting in MASSIVE-2 [Greenhalgh 96], MASSIVE-3 [Purbrick 00], NPSNET-V [Capps 00] and others [Joslin 04]. Looking at the publications of these systems, it shows already, that the research community, instead of focusing on new entire NVE systems, starts to spend more and more efforts on specific aspects, specifically





```
Face: se  At:  -2.0,  1.0,  2.5   Focus: table
┌----------------------------------------------------------┐
|                                                          |
|  c   c   c   c   c   c   d   d   d   c   c   c   c   c   |c: conference room
|                                                          |
|  c   .   +   .   .   .   d   *   d   .   .   .   +   .   |t: table
|                                                          |
|  c   .   +   .   .   .   d   d   d   .   .   .   +   .   |d: door
|                                                          |
|  c   *   *   *   *   *   *   *   *   *   *   *   *   *   |C: Chris
|                                                          |  O-> 1.0, O<- 1.0
|  c   .   +   .   .   .   .   *   .   .   .   .   +   .   |S: Steve
|                                                          |  O-> 1.0, O<- 1.0
|  c   .   +   .   .   .   .   D   .   .   .   .   +   .   |R: Rob
|                      @       |                           |  O-> 1.0, O<- 1.0
|  c   .   +   .   .   . \ t   t   t   .   .   .   +   .   |D: Dave
|                                                          |  O-> 1.0, O<- 1.0
|  c   .   +   .   .   .   t   *   t - S   .   .   +   .   |
|                          /                               |
|  c   +   +   +   +   + R t   t   t   +   +   +   +   +   |
|                              |                           |
|  c   .   +   .   .   .   .   C   .   .   .   .   +   .   |
|                                                          |
|  c   .   +   .   .   .   .   *   .   .   .   .   +   .   |
|                                                          |
|  c   .   +   .   .   .   .   *   .   .   .   .   +   .   |
└----------------------------------------------------------┘


Chris says we'll be using audio for most of the meeting today
Rob says fine

Say no problem█
```

Figure 2.3: The MASSIVE interfaces [Greenhalgh 95]: graphical (left) and textual (right).

enlargements with respect to numbers of users and environment size, improving graphics and supporting more hardware.

In the last decade or so, the military and gaming industry also started to join forces, creating games for military training and recruitment. MUDs started to evolve into full 3D large scale Massively Multiplayer Online Role Playing Games (MMORPGs), a subset of the more general Massively Multiplayer Online Games (MMOGs) that are usually in a phantasy or science fiction setting. Therein, advanced personalized 3D animated characters are used to represent the users and their actions. The environments are large scale detailed 3D environments. However, the number of interactive objects are still rather limited. Also, the animations are often just playbacks of fixed recorded animations, which result in repetitive motions, not very adaptive nor realistic. However, in the last years, Badawi [Badawi 06] argues that computer games have been using more and more physical realism in order to allow the user to push objects around or to blast them to pieces. *Rigid body simulation* and *ragdoll physics* have been providing games with more realism for quite a while, and with the uprise of physics Software Development Kits (SDKs) and hardware, the virtual environments in games are becoming more and more dynamic and more real-time simulations become possible, including fluids and gases. Figure 2.4 shows an example of how physical objects can be used in the gameplay. Still, while some basic physical actions are arising, we do support the view of Badawi [Badawi 06] stating that, apart from a few exceptions, more advanced interactions such as opening doors or realistically operating a machine remain unseen in computer games. When they are, they are ex-

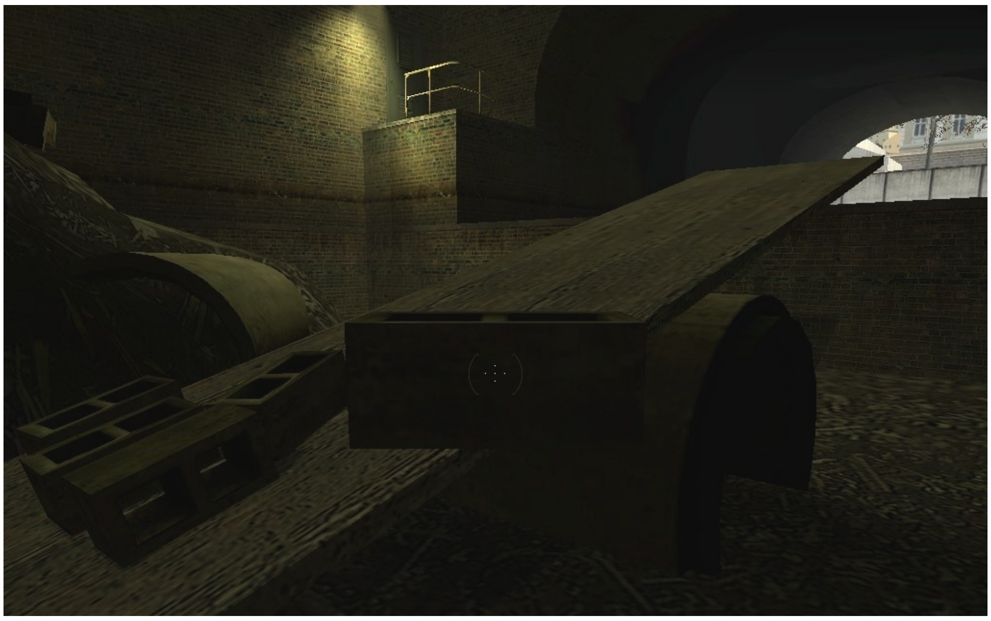


Figure 2.4: An image from *Half-Life® 2* [Half-life 2 07]. Rigid Body simulation is used in the gameplay. The concrete bricks are placed on the lever as a counterweight so the user can walk over the wooden board.

tremely simplified and e.g. picking up an object is done by teleporting the object to that user's virtual hand. We also believe that if an NVE user can interact more completely with the virtual world, his feeling of immersion would definately increase. In contrast to the gaming community, in professional VE areas, these kinds of interactions are often considered 'eye candy' and have therefore only been seldom applied.

An overview of past NVE systems is given in Figure 2.5. The interested reader can find more information and references to publications on NVE systems in [Macedonia 97, Joslin 04].

As NVEs also stem from single-user VR systems, it has also adopted many of the interfaces that were developed for these kinds of systems, one of the key areas in single user VR. Since Ivan Sutherland demonstrated the first Head Mounted Display (HMD), the Sword of Damocles in 1968, a whole range of VR interface systems have been developed. Differing display technologies (HMDs, workbenches, CAVEs, ...) have been employed with several input devices (gloves, 3D mouses, haptic arms, visual and electromagnetic trackers, ...) and then combined with different specifically designed interaction techniques.

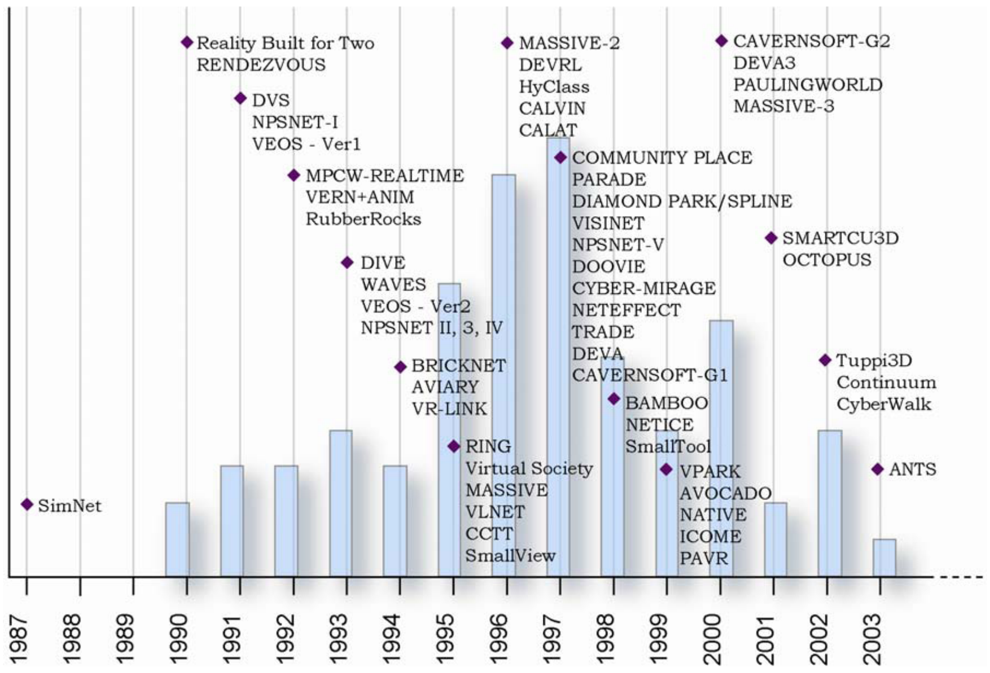


Figure 2.5: A historical overview of some of the most important NVE systems [Joslin 04]. The vertical axis represents the number of publications on these systems.

The main goal of most of these systems is to provide the user with a higher level of immersivity, the feeling of being there, through the use of new displays, input hardware or techniques. Giving a complete overview of all these systems is far beyond the range of this work, and although the potential of immersive interfaces is highly interesting, the reality is that it is unlikely that they will become available in every household in the near future, and the desktop computer will remain the common interface in the years to come. Therefore, in the remainder of this dissertation, as we are discussing interfaces, we will be focusing on desktop VE systems. On the other hand, the techniques discussed in the following chapters describing the interaction mechanisms for VEs, provide general solutions for all kinds of VE systems, irrespective of the interface. Of course, if links to other kinds of interfaces are relevant, they will be discussed as well.

2.2 User Embodiments

In order to graphically represent a user and the interactions he performs in the virtual world, to himself and to others, user embodiments are often utilized. These embodiments are in the VR community best known as *avatars*, a term stemming from Hindu mythology, where it is declared as: the descent of a deity to the earth in an incarnate form or some manifest shape; the incarnation of a god[2]. In [Thalmann 99] Thalmann states that in single-user VEs, avatars fulfill three distinct functionalities:

1. a visual embodiment of the user,
2. a means of interaction with the virtual world,
3. a means of sensing various attributes of the world.

For NVEs which can contain many users, avatars can be utilized for even a larger number of functionalities [Thalmann 01]. They are able to show:

1. if a users is present (perception),
2. where the users is (localization),
3. who it represents (identification),
4. where the user's attention field is (visualization of the other's interest focus),
5. what actions the user is performing (visualization of what others are doing),
6. the user's task or function (social representation).

It is thus obvious that avatars play a key role in NVEs, and have an important impact on the feeling of *shared presence*, the sense of being together [Thalmann 01].

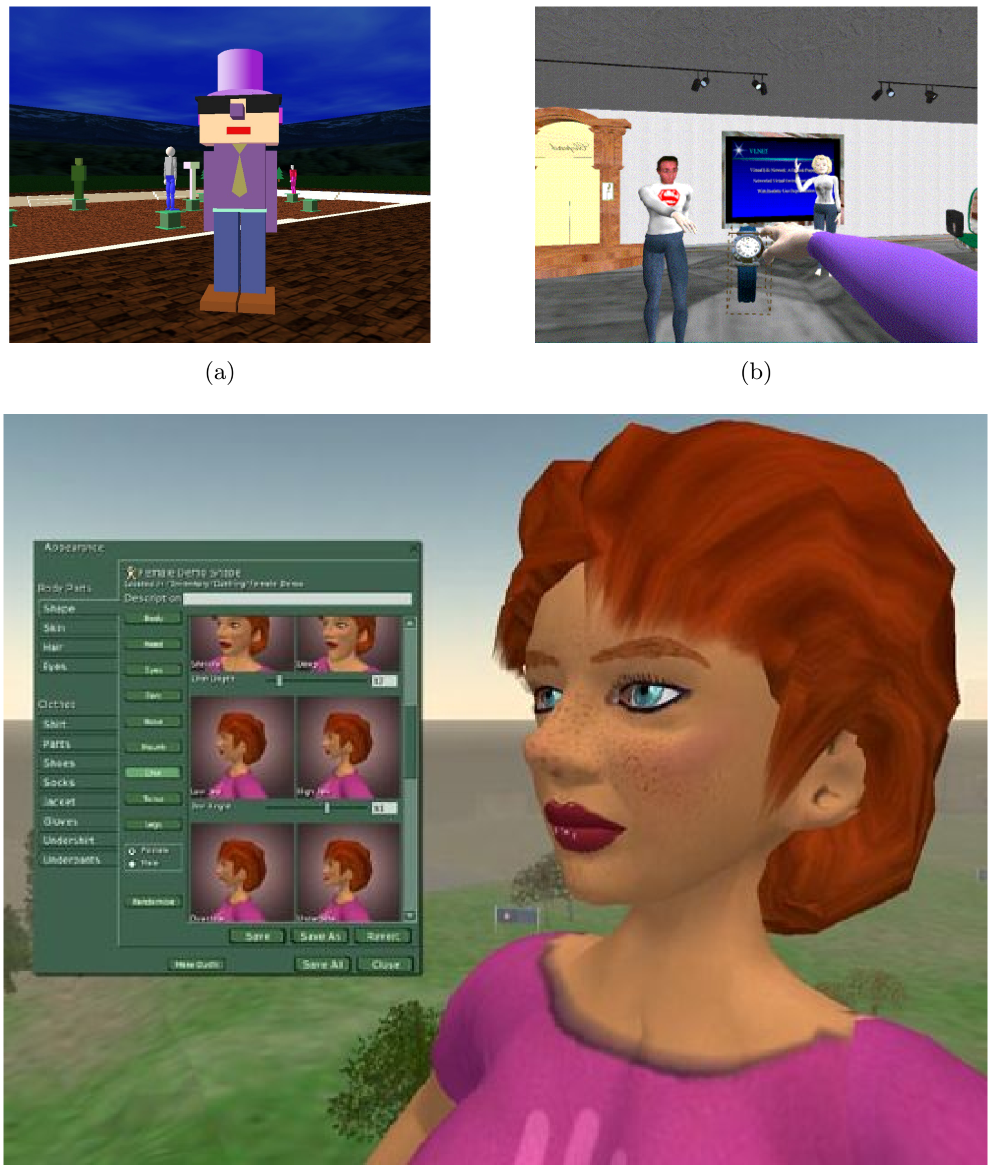
Many of the first NVEs used very simple avatars to represent connected users in the virtual world [Thalmann 00]. For example, RING [Funkhouser 95] utilized yellow spheres with green orientation vectors for user embodiment. Early versions of MASSIVE and DIVE, on the other hand, used so-called *blockies* to represent connected users [Benford 95]. Blockies are avatars that are only composed of a few very basic geometric shapes such as spheres and

[2]Wikipedia - Avatar (`http://en.wikipedia.org/wiki/Avatar`)

cubes. An example of a blockie avatar is shown in Figure 2.6(a). Although these early avatars already conveyed some interesting information about the user, such as his location and interest focus in the virtual world [Benford 95, Thalmann 99], they clearly lacked visual realism.

Articulated human-like avatars or so-called *virtual humans* (as shown in Fig. 2.6(b)) were introduced a few years later in the NPSNET system [Macedonia 95]. It soon became clear that integrating virtual humans in NVEs increased the natural interaction within these environments, and generally also resulted in a higher feeling of presence for connected users [Thalmann 99]. Furthermore, [Casanueva 01] demonstrates that when these virtual humans are able to perform animations, the feeling of presence is increased even further. As a result, it should not come as a surprise that almost all recent NVEs, MMOGs and other multiplayer games use animated human-like avatars.

Finally, in the last few years, with the uprise of MMORPGs and other NVEs that focus on virtual community building, such as Second Life [SecondLife 07] and There.com [There.com 07], user embodiment and character building have become crucial parts of the user experience. As a result, they offer their users very detailed and advanced avatars that are often also extremely customizable with respect to their visual appearance (see Fig. 2.6(c)). This allows users to create their own unique avatar, drastically increasing user identification with their avatar in the virtual environment. In combination with more natural interactions, the process of making avatars even more realistic and interactive remains an active area of research.



(c)

Figure 2.6: Avatar evolution: (a) blockie (DIVE [Hagsand 96]); (b) virtual humans (VLNET [VLNET 07]); (c) customizable avatar (Second Life [SecondLife 07]).

Part II

MODELING DYNAMIC VIRTUAL INTERACTIONS

Table of Contents

CHAPTER 3

Introduction

In the general introduction, we discussed that although interactive VR has been a research topic for several decades and a lot progress has been made in the fields of 3D graphics, animation and network distribution, we are still unable to create believable interactive virtual experiences. Furthermore, we discussed how this can at least partially be attributed to the lack of natural interactions with and within these environments. Mixing navigation and meaningful interactions with VR systems is therefore still a key research topic. However, this research often focuses too much on developing new ITs and input devices rather than being concerned with the underlying system which is concerned with how interactions can be represented, executed and altered. This is especially true for NVE systems which often consist of mostly static scenes.

Looking at the kinds of interactions (N)VEs allow, we find that most of them are limited to a few direct interaction techniques for selecting and moving objects and a simple way to navigate through a scene [Bowman 99]. Furthermore, Fraser [Fraser 00] investigated CSCW systems, concluding that these systems often take ad hoc approaches to object-focused interaction. In many systems, especially NVEs, interactions are mostly designed as a by-product of the development process [Smith 00]. In [Kallmann 01] the author, working in the field of autonomous avatars makes a similar conclusion, stating that simulation systems approach actor-object interactions by programming them specifically for each case. In addition, these poor solutions often only support a very basic animation system which is only able to display predefined animation sequences. Consider a basic example that illustrates this standard approach: when a user want to open an electronic door, he uses an input device to move his cursor to the button that needs to be pressed in order to open the door. When it is selected, the system starts by playing a predefined animation of the user's avatar, that moves its hand to the button and presses it. The 'press button' animation is then followed by another animation, opening the door, ending the interactions. When the task is repeated, the exact same actions and animations are reproduced. While this approach is common in VE systems, it is clearly different from how we interact in the real world. Furthermore, although this approach is simple, direct and fairly easy to implement, it is far from general and does not solve the problem for a wide range of cases. Finally, altering these interactions requires recompilation of the application's source code and/or off-line modeling of new animations, making run-time adjustments impossible.

As we already discussed in Chapter 1 of the previous Part, traditional systems often distinguish actors (avatars or agents) from other objects in the VE. However, virtual actors share many properties with virtual objects. Both need to be modeled, animated, and simulated. However, virtual humans usually incorporate more complex behaviors than other objects such as user or AI control, which is mostly the reason for this split. We, however, strongly believe that this distinction is unnecessary, especially if complexity is the basis for the distinction. Why not allow more complex actions for all entities in the VE? And where do you draw the line? Is an autonomous robot considered an actor or an object? An interesting view is given in [Badawi 06]:

> "the notion of object can seem pretty straightforward at first and according to the Oxford dictionary, an object is a material thing that can be seen and touched. This definition is extremely general and encompasses almost everything, but also everyone we see and touch on a daily basis. A human being is material and can be seen and touched, it is therefore, by definition, an object."

Another interesting point, in the context of interaction in NVEs, was given in [Broll 95] stating:

> "How different interaction requests are processed, as well as how many participants may interact with an object concurrently, is highly dependent on the object itself."

These observations, in cooperation with a *feature modeling* approach will form the basis of our general object interaction approach. Feature modeling is a technique that is mostly used for modeling products with CAD/CAM applications allowing the association of functional and engineering information with shape information [Bidarra 99]. In this approach, we aim to generalize all objects (including avatars and agents) in the VE and allow all these objects to interact with every other object by one general dynamic interaction mechanism. The information necessary for interaction is stored at object level instead of at application level.

The remainder of this part gives an overview of how this was realized. We begin by analyzing other approaches taken to solve this problem. Thereafter, we present the interactive object approach as a solution and present some examples and results. Finally we discuss some conclusions on the approach taken.

CHAPTER 4

Related Work

While modeling and describing virtual world objects is a very important part of the VE development process and many mechanisms for describing the visual elements of objects exist, only a few systems permit the dynamics and interaction properties of the objects to be described as well [Pettifer 99]. Most of these systems stem from the field of artificial intelligence (AI), where they are employed to inform autonomous agents how they can interact with the objects. Although this is not the our main goal, these systems provide some interesting aspects that will be useful for our purposes.

A first approach to solve the problem of how agents could be informed about object specific interaction information was proposed in [Levinson 96]. In this work, Levinson introduced an *Object Specific Reasoning* module, that created a relational table with geometric and functional classifications of objects. Furthermore, it kept some more interaction information on graspable positions for each object. This information could then be used to help interpret natural language instructions and to inform AI agents of an object's purpose and functionality. This approach was mainly applied to let agents grasp objects in their surroundings. While the approach has some interesting possibilities, it lacks the ability to create more complex interactive objects and does not solve the general interaction problem.

In the same field, a much more extensive system, was developed by Kallmann et al. [Kallmann 98] employing the ideas of feature modeling for the first time in the context of interactive VEs. In this work, the authors propose a framework for general interactions between virtual agents and objects. The idea is that all the information necessary for autonomous virtual agents to interact with an object is included in the object's description. For each object, interaction features and predefined plans are defined utilizing scripted commands. This so-called *Smart Object* approach is one of the most extensive systems describing all the functional and behavioral information of objects at object level. It employs interaction plans to synchronize movements of object parts with the agent's hand, and to model the more advanced functionality of objects. In this way, a *Smart Object* can instruct the autonomous agent on the actions to do step by step. Although this system provides the objects with a lot of interaction information, it is not general enough for our purposes since it still makes a distinction between different kinds of objects, and therefore requires different interaction schemes for different kinds of virtual objects (avatars, agents and other objects). Furthermore, agents should be able to interact with an object, the way they want to, not the way that is prescribed by the object modeler. *Smart Objects* prescribe all information, up to the joint positions and orientations of the agents hand that wants to interact with it, leaving no space for an agent's own interpretation. Also, even though animations are adjusted in real-time using inverse kinematics (IK), an agent in the same position performing an action, will always move in the exact same manner, positioning his hand in the exact same position when interacting with the same object. Unfortunately, the works describing the approach do not reveal if and how object-actor or actor-actor objects are supported. Finally, since this approach aims at human-like agents only, it is not suitable when working with different kinds of agents. With this system, it would be necessary for each object to have a different plan of interaction for every (kind of) agent. Thus, although the idea of feature modeling of interactive properties for VR objects is very promising and works perfectly for its purposes, the approach taken here will not be general enough to suit our purposes of more dynamic and interactive VEs.

This view is also shared by Badawi et al. [Badawi 06], who implemented another approach into a system wherein autonomous agents interact with the VE through *Synoptic Objects*. The information stored within these objects form a general description of the interaction process, without being specific, a kind of interaction synopsis, as they refer to it (hence the name). In this approach, *interactive surfaces* describe areas on the object that take part in the interaction and the space around the object that is affected by the interaction (influence surface). Furthermore, an object describes the interactions it can undergo through a set of basic actions which tell the agent what actions it needs to perform on the object in order to fulfill a task. How these actions are performed is left to the agents themselves, so there is only a loose coupling and, in contrast to the Smart Object approach, objects do not need to maintain information on different kinds of actors and thus do not determine the interaction process. Complex actions are created by combining interactive surfaces and basic actions through the use of finite state machines, which indicate the sequence of actions to perform during a specific interaction with an object. Since the geometric information is only composed of surfaces, without being overly precise, it allows the agent to adapt its animation to the part of the surface that is most suitable to its current situation, and thus avoids having the same repetitive robotic animation for the same interaction. Seven basic actions were described, covering most kinds of interactions. These always take an actor and an object without making any assumption about the nature of the actor. The disassociation of form and function allows all objects with the same functionality to share the same complex action and preserves that functionality even when e.g an object's geometry is changed. This approach solves some of the Smart Object's issues such as taking over the actor completely during interaction and the resulting problem with different types of actors. However, this solution is as a result of its goals too focused on agent-object interactions to provide a more general solution for all virtual world interactions. For example the basic assumption that actions take an actor and an object limits this system to these types interactions. Object-object and object-actor interactions are not intrinsically supported. Generalizing the approach would require the definition of an action for every possible inter-object interaction. Furthermore, the presented system does not support more than one interacting agent.

Thus, while several interesting approaches have been devised in the past, to our knowledge, no dynamic interaction mechanism that allows interaction between every possible kind of VE object to interact with every other object exists. In the following chapter we will discuss our interactive object system. We will first give a brief description of the overall system and then we will present the details of object representation, application in an NVE system and end with describing some results.

CHAPTER 5

A Dynamic Interactive Object System

5.1 Dynamic Interactions

Our aim is to create a platform that allows developers to create totally dynamic virtual worlds in which every object can interact with every other object, making no distinction at interaction level between ‘plain objects’, avatars or agents. Furthermore, we want interactions to be run-time adjustable and application independent.

In order to meet these goals, we employ a feature modeling approach, starting with VE objects that, apart from a visual representation, also contain their interaction information and behavior. Then, in order to create dynamic VEs, we devised a format that is able to describe VE scenes that are constructed of these *interactive objects*. Thirdly, we developed an interactive platform, the *interaction layer*, that can easily be integrated in applications and is able to simulate the dynamic virtual scenes by exploiting the interaction information that is provided with these object descriptions. This platform uses the concept of *interaction links* to provide a way to allow communication between objects. These channels allow objects to call each other’s behaviors and communicate the actions and behaviors they are performing. Finally, in order to provide a link between actors (human or AI) that wish to control an object, and the controlled objects, we created the possibility to construct *object controllers.* These allow an application developer to implement interaction techniques or can be used as the interface for an AI system to control an intelligent object by sending *commands* directly to it.

This summarizes the basics of how the overall system works, in the next sections, we will discuss the different aspects of our interactive object platform in more details. Thereafter, some examples will be given.

5.2 Interactive Objects and Dynamic Worlds

In order to cover all aspects of objects, that are required for general interactions in interactive VEs, we identified three distinct groups of object features. First of all, we have the basic *object properties*, that depict what the object looks like, which parts it consists of and how these parts relate to one another. Secondly, the object has a set of *interaction properties* depicting the object's external interface, its interactivity features and functionality. Finally, an object describes its *behaviors* in order to define how it will react on certain interactive events and requests.

Obviously, this overall structure is similar to the description of the Smart Objects we discussed in Chapter 4. This might not come as a surprise, as our approach needs to describe similar object information. However, whereas the Smart Object approach was designed for VR simulations wherein AI actors use high-level planning to achieve their goals, focusing on actor-object interactions, we take a much broader approach. We aim at generalizing all objects and allowing every object to interact with every other object in interactive VR applications. Also, in the Smart Object approach the objects contain information on how AI actors need to interact with them. In our approach, we maintain generality by leaving out all information that is not directly related to the object itself. We did this because, as we explained in the previous chapter, we do not believe that this information is relevant for the object itself but rather for the agents, users or other objects that want to interact with it. If AI actors want to interact with an object (or actor), it is their responsibility to know or to find out how to interact with that object, not the responsibility of the objects, just like in the real world. Consider an application with many different kinds of actors. Then, the object description would have to contain interaction information for every different kind of actor and even for every object in the VE, since we want every object to be able to interact with every other object. This would mean an enormous amount of interaction informa-

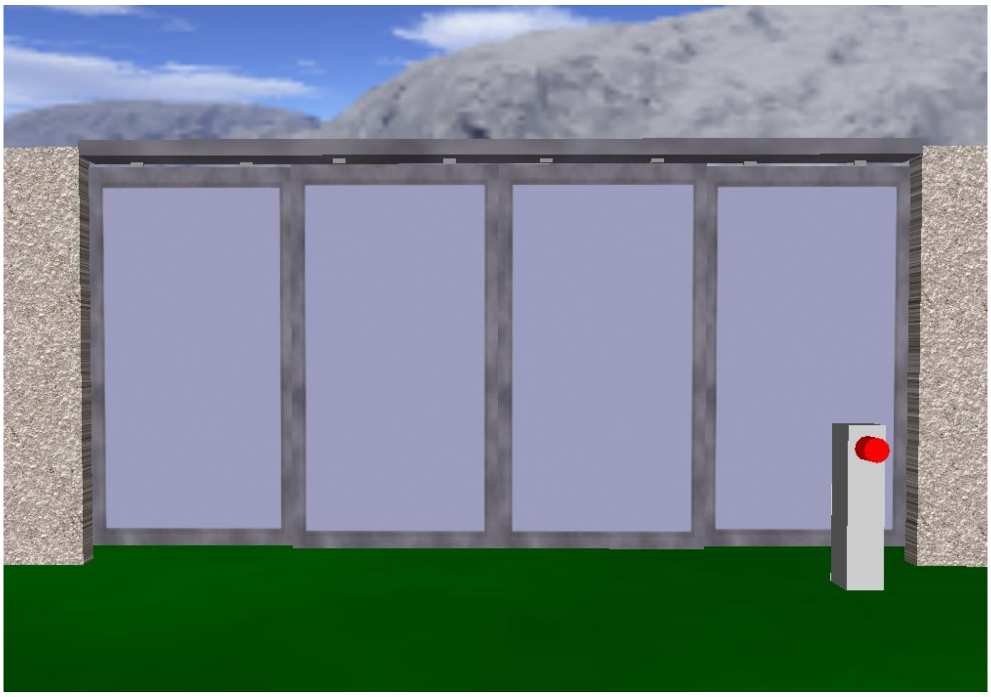


Figure 5.1: An example of an interactive door object with two movable sliding parts and a button triggering the doors to open and close.

tion. Also, the introduction of new objects in the environment would mean an enormous amount of work, since all other objects used in the application would have to be adjusted in order to know how to interact with the new object. Finally, in contrast to the *Smart Object* approach, we make no distinction between actors and other objects. Subsequently, every object/actor can interact with every other object/actor using one single interaction scheme where all are handled equally and actor-object coupling is much looser.

We will now discuss the three interactive object property types in more detail. Throughout the discussion, we will refer to an example object describing a button-controlled sliding door that is shown in Figure 5.1. In order to further clarify the properties, we will give some partial object descriptions of the example as well. The complete door example description is given in Appendix C.1.

5.2.1 Object Properties

This part of the description comprises a full description of how the object is constructed, what states it can be in and its internal basic actions. In order to be identifiable, every object has a name and an identifier. Furthermore, a text based description can be added in order to provide a more extensive explanation of the object and its purpose. The object properties thus include the object's parts containing:

- a graphical description (3D model);
- its localization in object space (position and orientation relative to object space);
- a collision object description (if different from the part's 3D model);
- its movement constraints (if any, relative to object space).

The object properties also describe the object's possible actions. Actions are defined as combinations of object part movements or movements of the entire object itself over a certain amount of time.

Lastly, the object properties make it possible to define state variables of several basic types (string, boolean, float, integer,. . .), with a name and an initial value. The actions and variables can then later be used as building blocks for the behaviors that can be triggered by some other object or a controller by sending the right commands to the objects as will be discussed in section 5.2.3.

Regarding the door object example, this is constructed of several different *parts*:

- the door frame, a rail that is fixed (constraint, to disable movement);
- two fixed outer panels (identically constraint);
- two sliding door panels (able to move in one direction for a limited distance);
- two posts, one on each side of the door (also fixed);
- two buttons that need to be pressed in order to open/close the door.

The actions that are defined consist of the possible moves of the object including: open and close actions of the left sliding panel (by translating the left sliding panel) and similar actions for the right panel. Furthermore, two boolean state variables are defined, one to store the 'open' state and one to check if the open/close actions are being processed (doorsMoving). Listing 5.1 consists of a part of the example's object properties. It includes all different aspects, however not all parts and actions are included. Two different kinds of parts are described, one non-fixed specifying how it is constrained and one fixed object that cannot move in any way. Both specify their own collision box. The movement actions for the left sliding door and the object state variables are also included.

Listing 5.1: Partial object properties description of the interactive door example

```
<OBJECT_PROPERTIES>
    <DESCRIPTION>
        This is the interactive object description for a button controlled sliding door.
        The door can be opened and closed by collision triggers coupled to the buttons.
    </DESCRIPTION>

    <PART partid="glassdoor_left" filename="glassdoor.ms3d" parentid="doorframe">
      <POSITION x="-2.9" y="1.85" z="0.1" />
      <ORIENTATION x="0" y="0" z="0" />
      <COLLISIONBOX
          xsize="2" ysize="3.8" zsize="0.1"
          xpos="0" ypos="0" zpos="0"
          xrot="0" yrot="0" zrot="0" />
      <PART_CONSTRAINTS>
        <MAX_TRANSLATE_CONSTRAINT upx="2.0" upy="0" upz="0"
                                  lowerx="0" lowery="0" lowerz="0" />
        <MAX_ROTATE_CONSTRAINT clockwisex="0" clockwisey="0" clockwisez="0"
                               cclockwisex="0" cclockwisey="0" cclockwisez="0" />
      </PART_CONSTRAINTS>
    </PART>

    <PART partid="buttonfront" filename="buttonsphere.ms3d"
          parentid="buttonbase1" isfixed="TRUE" >
      <POSITION x="2.5" y="1.2" z="1.2" />
      <ORIENTATION x="0" y="0" z="0" />
      <COLLISIONBOX
          xsize="0.2" ysize="0.2" zsize="0.2"
          xpos="0" ypos="0" zpos="0"
          xrot="0" yrot="0" zrot="0" />
    </PART>

    <ACTION name="open_left_door">
      <TRANSLATE_PART partid="glassdoor_left" x="-2" y="0" z="0" time="2000" />
    </ACTION>

    <ACTION name="close_left_door">
      <TRANSLATE_PART partid="glassdoor_left" x="2" y="0" z="0" time="2000" />
    </ACTION>

    <VARIABLE type="bool" name="isClosed" value="false" />
    <VARIABLE type="bool" name="doorsMoving" value="false" />
</OBJECT_PROPERTIES>
```

5.2.2 Interaction Properties

The second set of features that is contained in the object description consist of the *interaction properties.* This part is where the actual interface and the interactive parts are described. Firstly, the object modeler can define *object* *commands*. These form the interface for controllers and other objects that wish to utilize the object and start one of its actions or behaviors. Object commands have a name and can have any number of parameters (defined as comma separated strings). Secondly, *interaction zones* can be described. These zones specify which areas take part in the interaction process. They can consist of an object's part, an entire object, a basic shape like a box, sphere or cylinder region or a 3D modeled shape at a specified position relative to the object. Finally, the object's *interaction triggers* can be defined. This version supports three different kinds of triggers: *collision triggers*, *proximity triggers* and *timed triggers*. The interactive door has one object command: `MoveDoors` and one collision trigger consisting of the two buttons. The interaction properties description is shown in Listing 5.2.

Listing 5.2: Interaction properties of the interactive door example

```
<INTERACTION_PROPERTIES>
  <OBJECT_COMMAND command="MoveDoors" />

  <INTERACTION_ZONE zone_name="button1zone">
    <PART_REGION regionid="frontbuttonregion" partid="button1" />
  </INTERACTION_ZONE>

  <INTERACTION_ZONE zone_name="button2zone">
    <PART_REGION regionid="backbuttonregion" partid="button2" />
  </INTERACTION_ZONE>

  <TRIGGERS>
    <ZONETRIGGER triggerid="door_trigger" zones="button1zone,␣button2zone" />
  </TRIGGERS>
</INTERACTION_PROPERTIES>
```

Triggers can be used to invoke actions as well as behaviors. How an object reacts to a flagged interaction trigger or to a received object command is discussed in the next section. The interaction mechanism itself is explained in section 5.3.

5.2.3 Object Behaviors

The *object behaviors* can be used to describe how an object reacts to certain events and interactions in the virtual world. The biggest part of this description consists of the behaviors themselves, that are specified in a scripting language. Scripting languages are programming languages that typically do not require to be compiled in advance, but are interpreted at run-time. As a result, scripts can easily be altered at run-time, which is a major step toward achieving more dynamic environments. In this version, we devised our own scripting engine supporting its own language. This language has, apart from standard programming language statements, support for adapting and requesting states from object variables, controlling object actions and animations, allowing the object to communicate with other objects and triggering other behaviors. Furthermore, application developers can implement their own functions in C++ and provide hooks to them in the scripting language, making them very powerful and adaptable in describing behaviors. Also, since scripts can be altered at run-time, an object's behavior can be replaced or altered during simulation, resulting in different reactions for specified interaction events, providing VE designers with a very dynamic interaction mechanism. Also, an object modeler can specify how many times the script can be executed (if not indefinitely) and how many executions of this behavior can run at the same time. This will especially be useful when behaviors that can be triggered by a limited number of users in a collaborative setting.

Apart from behaviors, this part of the object features is concerned with defining the coupling between the interactive object's interface (commands and triggers), object actions and the scripted behaviors. Two types can be specified: couplings between object commands and behaviors/actions and couplings between triggers and object commands/actions. In this way, the designer can define the behavior that is started when the object receives a command from another object or controller, or when for example a collision trigger is activated. How this mechanism works in more detail is explained in section 5.3.

Listing 5.3 gives the object behaviors of the interactive door. One scripted behavior is defined (`MoveDoorScript`). It opens the doors when closed and vice versa. It has no restrictions on how many times it can be called but can only be run once at a time. To make sure it is not running twice, the script first checks if the doors are already moving (by checking the state variable). If the doors are already moving, e.g. another element has already triggered this behavior, the script ends. Otherwise, the script checks if the doors are open or closed. If the doors are closed/open, the interactive object actions (that were described in the object properties) to open/close the two sliding panels executed after setting the state variable `doorsMoving` to TRUE. When these tasks are finished, the door's open state variable is changed and the `doorsMoving` state is set back to false whereafter the script finishes. Finally, the behavior features describe that both the zonetrigger (collision with one of the buttons) and previously defined `MoveDoors` command result in the execution of this behavior.

Listing 5.3: Object behaviors of the interactive door example

```
<OBJECT_BEHAVIORS>
  <SCRIPT name="MoveDoorsScript" maxsimultaneousexecutions="1" maxexecutions="unlimited">
      push_object_var doorsMoving     <!-- if doors are already moving end -->
      jump_if_false 3
      end
      set_object_var doorsMoving 1    <!-- else set doorsMoving true -->
      push_object_var isClosed        <!-- if doors are closed goto open commands -->
      jump_if_true 13
      pusharg_const close_left_door   <!-- call actions to close both doors -->
      call_command performAction
      pusharg_const close_right_door
      call_command performAction
      set_object_var isClosed 1       <!-- set doorsClosed variable true -->
      set_object_var doorsMoving 0    <!-- set doorsMoving variable false -->
      end                             <!-- end -->

      set_object_var doorsMoving 1    <!-- similar for opening -->
      pusharg_const open_left_door
      call_command performAction
      pusharg_const open_right_door
      call_command performAction
      set_object_var isClosed 0
      set_object_var doorsMoving 0
      end
  </SCRIPT>

  <TRIGGERCOMMAND triggerid="door_trigger" commandname="MoveDoors" />

  <COMMANDSCRIPT commandname="MoveDoors" scriptname="MoveDoorsScript" />
</OBJECT_BEHAVIORS>
```

5.2.4 The interactive Object and World Description Formats

As you may have noticed from the code listings, in contrast to the Smart Object approach, that uses a proprietary textual description for objects, our interactive object approach utilizes XML-based files. The major advantages of XML include readability (it is text-based), easy validation (automatically, using a well specified Document Type Definition (DTD), standard parsing tools (SAX and DOM) and the ability to be streamed. Furthermore XML can be compressed and decompressed quite efficiently, which makes it very suitable for network distribution. These aspects are however not the subject of this work and will therefore not be discussed further.

The world description, on the other hand, is concerned with global simulation parameters and defines how the scene is constructed. We will however not discuss the entire formats in detail. An example scene containing a few interactive objects is described in Listing 5.4.

Listing 5.4: An example interactive object scene: a room with sliding door entrance and a table in the center

```
<CIWORLD worldname="testworld" sizeX="100" sizeZ="100">

  <!-- ground plane -->
  <INTERACTIVEOBJECT filename="floor.cio"
    xpos="0" ypos="0" zpos="0"
    xrot="0" yrot="0.0" zrot="0"  />

  <!-- Sliding doors for the room-->
  <INTERACTIVEOBJECT filename="slidingdoor.cio"
    xpos="0" ypos="0" zpos="-20"
    xrot="0" yrot="0" zrot="0">

  <!-- Right wall -->
  <INTERACTIVEOBJECT filename="wall.cio"
    xpos="20" ypos="0" zpos="-30"
    xrot="0" yrot="90" zrot="0"  />

  <!-- Left wall -->
  <INTERACTIVEOBJECT filename="wall.cio"
    xpos="20" ypos="0" zpos="-30"
    xrot="0" yrot="90" zrot="0"  />

  <!-- Back wall -->
  <INTERACTIVEOBJECT filename="doublewall.cio"
    xpos="0" ypos="0" zpos="-40"
    xrot="0" yrot="0" zrot="0"  />

  <!-- Table in the center -->
  <INTERACTIVEOBJECT filename="table.cio"
    xpos="5" ypos="0" zpos="-30"
    xrot="0" yrot="0.0" zrot="0" />

</CIWORLD>
```

5.3 The Interaction Layer

Now that we have constructed a description for interactive scenes and dynamic objects, we require a system to integrated them in VE systems. The software component that we developed in order to support and simulate dynamic virtual worlds wherein every object in the interaction cycle is an interactive object is the *interaction layer*. It is the central control unit for all object interactions in the VEs that wish to employ the interactive object approach. The tasks that it needs to perform include:

1. constructing a dynamic interactive virtual environment from a world description;

2. simulating the movements of the objects and check collisions;

3. creating communication channels for objects to support and mediate interactions;

4. checking object trigger states;

5. executing behaviors and scripts.

These tasks and their workings will now be discussed in more detail.

Task 1: Creating a dynamic interactive virtual world

The process of constructing a dynamic VE is fairly simple. The interaction layer starts with loading an interactive world file and setting the global scene parameters. Then, the scene loads all interactive object descriptions creating the core objects. As soon as this is done, the interactive object instances are created from their cores and placed at the specified location in the scene. Then the virtual world is ready to start the simulation.

Task 2: Simulation

In this first version of the interactive object platform real-time physics are not integrated. In the following Part we will describe an extended version which does. As a result, here the interaction layer is also responsible for calculating object movements. This is done by the simulation component which allows for some very basic object movements. If an objects is not in collision, it can be translated and rotated every timestep. However, in this basic version no gravity is implemented, neither are the concept of speed and friction. The collision component is responsible for checking object collisions every timestep. we utilize the Swift++ collision detection package (Speedy Walking via Improved Feature Testing for non-convex Objects [Swift++ 07]) to calculate object collisions. When an object or part requests a move, the simulation component of the interaction layer performs this request taking into account its specified rotational and translational constraints. When two or more objects or parts are colliding, and one wants to move in the direction of another object, all objects in the colliding cluster receive a move request and perform that move, unless one of them is restricted due to constraints, in which case none of the objects are moved (or not in every direction). Although this approach is not physically correct, it suffices for the moment, as we are mainly concerned with creating more dynamic interaction possibilities. Furthermore, due to its simplicity the approach is very lightweight, making it possible on less powerful systems as well and is independent of third party software. In the next part of this work, we will discuss how a rigid body simulator results in a much more realistic solution, however requiring more resources.

Task 3: Interactive Links for Communication

In order to create communication channels for the objects to mediate interactions, we employ interactive links. These links can be used to call each other's behaviors and communicate the actions and behaviors they are performing to others. The most important links that the interaction layer controls are: contact links, attachment links, controller links and parent/child links.

Contact links are set up when two or more objects are in contact with each other. The interaction layer creates them when the simulation component receives a collision report from its collision detection component for distributing movement messages. *Attachment links* on the other hand, are constructed between parts of different objects that somehow need to be attached to each other. One of the most obvious examples is when a user grasps an object. Since no physical simulation engine is present, attached objects communicate their movements through these links, similarly as with contact links, however these links do not disappear when the collision would end. In this way, if an the virtual hand would move in a direction away from the object, the object will still follow this movement. The third kind of links, *controller links* are set up by interaction layer when an object controller is initialized by the application. These links provide a way for the application to send commands to objects directly. An *object controller* can have links to several different objects and can send single messages to just one of them or to all of them at the same time. Furthermore, when an object wants to control another object these links will also be used in a similar way. Lastly, *parent/child links* are automatically set up when an object becomes a parent of another object, or when this is specified in its description.

Task 4: Checking Triggers

At every time step, the interaction layer checks the state of every object trigger. More specifically, this requires collision detection, which is done already by the simulation component, in order to activate collision triggers. Also, the distance between objects needs to be checked in order to see if any proximity triggers should be activated. The collision component does also support this. Finally, the timing component needs to check if any timed trigger should be activated when its time has elapsed, or the activation time has been reached. If a trigger is activated, the connected actions and/or behaviors are executed. Actions can be called immediately, as they only consist of simple predefined movements and are executed by the interaction simulation controller. Behaviors, as explained earlier, are defined in a scripting language and need to be interpreted and executed at runtime.

Task 5: Scripting and Behaviors

As explained earlier, every object can specify its own interface in its description through commands and behaviors. These commands can be used to trigger an object behavior. Commands can be sent by controllers or by other objects through the links. Some commands result in default behaviors, such as: `onContact, onEndContact, onMove, move, onAttach, Detach`, ... The default behavior can easily be overridden by implementing the command and resulting behavior in the object description. An object's interface can be requested by other objects or by controllers via the *object management* component, allowing other objects to know what commands are implemented. When a command is received by an object, the behavior or action that it is coupled to is executed. If it does not concern a default behavior and the command is not implemented, the command is conveyed to all (if any) parent links of the object, in case that one of them can perform the necessary task. In our experience, most objects, however, have no or at most one parent objects.

In order to allow run-time adaptable behaviors, we implemented a specific scripting engine for this purpose that, apart from basic programming statements, has hooks to functionality that is required by the interactive object system such as:

- checking and changing object variables;
- executing object commands and behaviors;
- starting and stopping predefined animations;
- change object features (add/remove/change: parts, variables, triggers, commands and behaviors).

These features are extremely powerful, as it allows every aspect of an object to be adapted at runtime. Furthermore, an application developer is free to add scripting hooks with his own, application specific, functionality. When started, every behavior is run in a separate thread.

This interaction layer can easily be integrated in any application as a part of the simulation component. Figure 5.2 shows how the interaction layer, interactive object instances and controllers relate to application components.

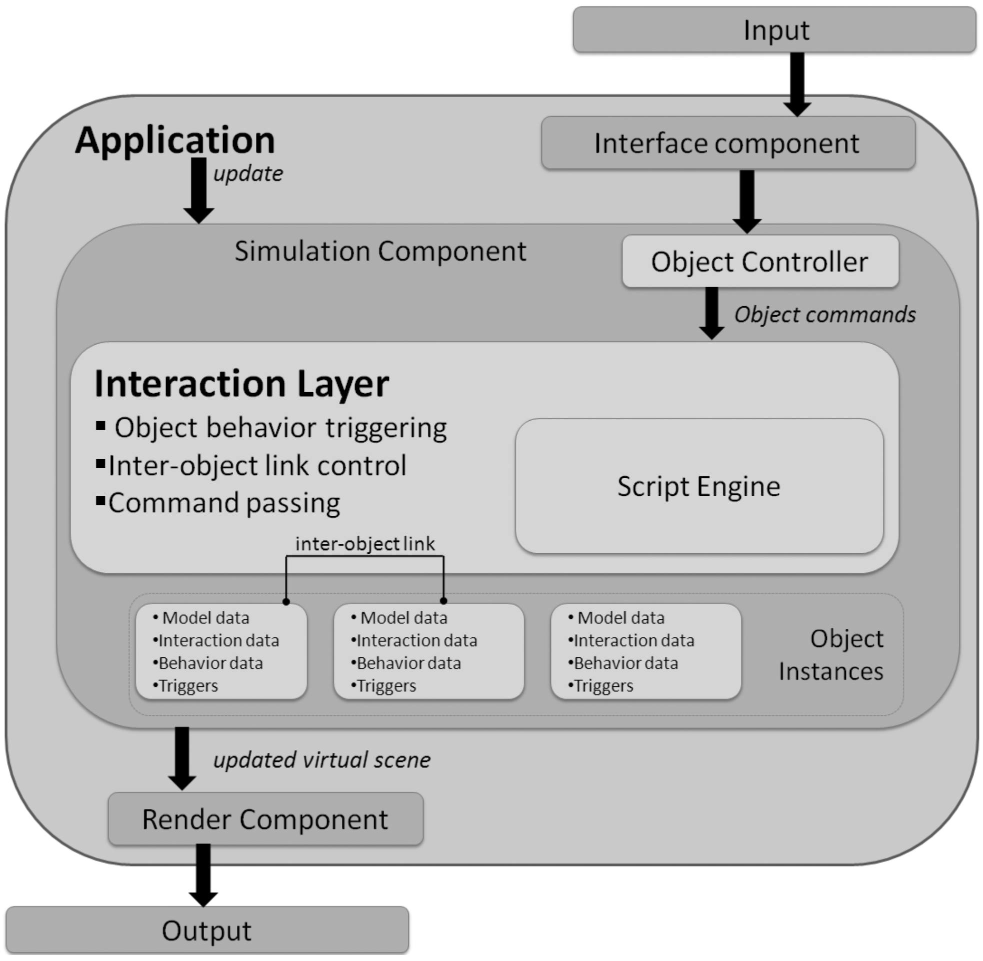


Figure 5.2: The interactive object approach. An application can include the interaction layer that loads and simulates interactive object worlds as part of the simulation component. Object controllers can be implemented to convert input into interactive object commands.

5.4 Network Distribution and Simulation for NVEs

To allow remote users to simultaneously interact and collaborate within a NVE, they must have a consistent view of the shared world and therefore require to be informed on every relevant state change of that VE. The simulation component of the application can realize this in combination with the networking component. The interactive object interaction mechanism discussed in the previous sections, was designed to work for all kinds of VE systems, including NVEs. However, a fixed built-in distribution architecture is not a part of the approach. This was deliberately not provided in order to remain independent of the architectures and applications it could be used in. As discussed in Chapter 1 the choice of how data is distributed in an NVE does not only depend on the data that needs to be distributed, but is also largely dependent on the underlying network and the application requirements. However, to show how this could be realized, we present some distribution setups. As a first distribution architecture, we propose a client/server architecture with a centralized simulation, wherein the server acts as a simulation manager. This means that the server is responsible for loading and creating the virtual world, simulating it, processing updates from the clients and distributing updates to all clients. With respect to virtual world simulation, this means that the server's interaction layer is responsible for all movements, collisions and interpretation and execution of scripted behaviors. As a result, the client's systems are alleviated from these rather resource intensive tasks and are only responsible for loading and rendering the graphical aspects of the VE and converting input into interaction requests. Their interaction layer is thus only responsible for the controllers that convert input into interaction requests, that are sent to the simulation controller. As a result, much more complex scenes can be simulated on less powerful machines and far less memory is required since the interaction information does not need to be loaded. The server handles these interaction requests on a first-come-first-serve basis.

Apart from running the simulation and handling the interaction requests, the server is also responsible for the initialization of all new clients by sending them a consistent state. For the actual distribution of the data, we use a hybrid approach. At startup, clients connect to the server with reliable TCP to receive the current state of the virtual world. The system thereafter falls back onto UDP to keep the different clients synchronized. Furthermore, to reduce network load, the server sends state updates to all interested clients at once using multicasting. Multicast transmission delivers source traffic to multiple receivers without adding any additional burden on the source or the receivers while using the least amount of bandwidth of any competing network technology. An overview of this network setup is given in Fig. 5.3.

To reduce the network load even further, the server exploits information from the interaction layer and object descriptions to see which object parts have changed state and must therefore be communicated to clients. Furthermore, object constraints are also exploited in order not to send information on parts that are fixed, or that can only rotate (so no translations updates need to be transmitted for these objects). Since UDP is an unreliable protocol, we do not rely on incremental updates, but always send full positions and orientations. Optionally, TCP *keep-alive* messages containing positional data can also regularly be sent to ensure at least some synchronization under all circumstances. If in the meanwhile a more recent UDP message has been received by the client, the (outdated) TCP message will be ignored. Since we work at high update rates (25 per second), our approach is relatively robust against small percentages of packet loss. The TCP messages will only be required in situations wherein long sequences of UDP messages, concerning the same object/part are lost.

We must note that the network messages sent here only contain transformational object data (positions and orientations). If an update message from the server gets lost, there is only a synchronization issue at those clients that did not receive it, and only for a short period until they receive the next update of that object by UDP or TCP. If a client's interaction request message, for example for a move forward, is lost on its way to the server, no adjustments will be done at the server running the simulation, and thus no move forward will happen on any client, hereby maintaining its consistency. This might however cause problems, since the user might believe that his interaction is requested. This could be resolved by sending the update requests with a reliable protocol, but this could then on its turn increase the round-trip delay time for the interaction (as discussed in Chapter 1). Another benefit of using this centralized server is that we can have persistent worlds that do not disappear or lose state when all clients disconnect, as is the case with some P2P solutions. Consequently, as long as the server is running, the world keeps existing and evolving. Logically, our CS approach suffers the same advantages and disadvantages of other systems as discussed in Section 5.5.

Other approaches are just as well possible. For example, we can use the same simulation distribution approach but on top of a P2P network architecture. Then, one client e.g. the first client that enters the world or the most powerful, could be selected to act as the simulation node. This client's computer would then act as the server. When he logs off, the next node in the

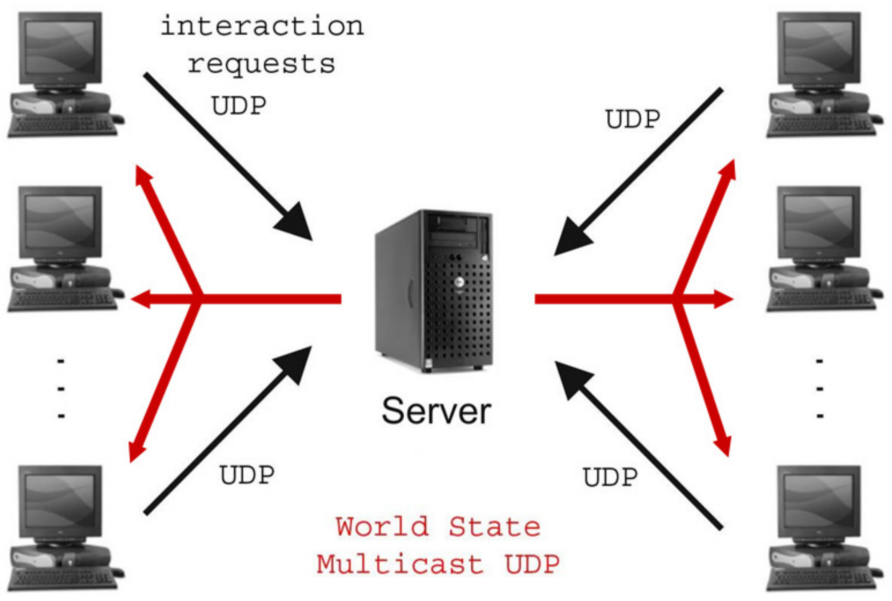


Figure 5.3: Network setup: after initialization using TCP, the clients send their interaction requests using UDP. State updates are distributed by the server using multicast.

selection procedure will take over the simulation and so on. A distributed simulation approach is also a possibility wherein different clients simulate different parts of the virtual world simulation. As results, the application's simulation component would need to implement a mechanism for splitting up the simulation onto the different clients. For demonstrating our technique we took the single simulation controller CS approach. The results are presented in the next Chapter.

5.5 Evaluation of the Interactive Object approach

5.5.1 A Simple Controller Example

We already discussed an example of an interactive object in the previous chapter, but did not give an example of how input can be employed to control such an interactive object. As an example, we describe an object that can be used to navigate through an interactive world and the controller that converts keyboard input into object commands. The interactive object itself consists of one part e.g. an avatar's collision box, actions for rotating and translating it and commands that can form an interface for these movements. The object behaviors are limited to the coupling of the commands to the movement actions.

Once this object is created, the application developer can create a controller. By specifying this object as the controlled entity, the interaction layer will automatically generate a controller link to the object when the controller is initialized. Now all the application developer needs to do in order to manipulate the object, is send the correct commands to the object through the link. As an example, keyboard presses could be used to control the object by mapping keyDown events onto the command `moveBackwards`. A keyUp can similarly be translated into a command such as `moveForward` and a keyArrowLeft could trigger the `rotateY` command with some specified number of degrees as a parameter. Upon receiving this command, the object will request execution of these actions to the simulation component.

5.5.2 Discussion of Examples

Both the interactive door object from the previous chapter and the controlled object example we have just discussed show the basics of how interactive objects can be constructed and controlled. We must hereby note that due to the freedom that our approach provides, related to how objects and controllers are built, the same object can often be created in several ways. As an example, consider the door object. A modeler could decide to model the doors and panels with a modeling package as one object containing predefined open and close animations. The resulting object description would then only have this door object and the posts and buttons as parts. The script could then just check if the doors are moving, or if one of the animations is active or not and play the predefined open/close animation. Of course, if afterward some other behavior would need to be specified, e.g. opening only a single door panel, the modeler would need to model this animation using the modeling package again, which would not be the case with our described object.

It is these kinds of design choices that will need to be made when new objects are created, and poor choices can lead to an object that is less adaptive. However, it would still be an improvement over the traditional approach in which every interaction is implemented hard coded into the application. On the other hand, if the right choices are made, the adaptivity of our interactive objects is practically unlimited as every aspect of it can be changed at run-time. As a relatively minor downside, object controllers still need to be hard coded into the application. However, their only task is to send commands to the objects through the interaction layer, and they are often closely coupled to the application. In future work we could allow controllers to be implemented in scripts as well, making the applications even more adaptable. Furthermore, objects controlling other objects can be added at any time. As an example consider a remote control for the door. A small object with a single button can be created, similar as the post with the button, except for the movement constraints. This object could be added as a child object of the original door object. Then this object could define a `zonetrigger` and couple it to the same `MoveDoors` command as was done with the original post buttons, and the remote could be introduced into the environment at runtime without any modifications to the original object. As the `MoveDoors` command is not implemented by this object, it will be passed to its parent, the original door object, and executed there.

Finally, we must note that although the design and modeling of our objects with its interaction features might seem more complex and time-consuming than the traditional approach (hard coding interactions), once modeled objects are highly reusable. First of all, entire objects can be reused in other applications. Secondly, similar objects, with a different look could reuse state variables, behaviors and constraints but replace the graphical components or models. Finally, some common behaviors can be used in by several other objects as well. Consider the `MoveDoorScript` behavior, with slight changes (other animations, move actions), it could be used for e.g. cupboards, windows, a CD rom drive, ... Furthermore, the generalization of object types results in the fact that every object can trivially interact with every other object without the specific need to alter the application.

5.5.3 Evaluating Interactive Objects in an Existing NVE

In order to apply our approach and distribution architecture in the field, we integrated our interaction mechanism in the ALVIC (Architecture for Large-Scale Virtual Interactive Communities) platform described in [Quax 03] and [Quax 07]. This system was originally developed to allow the creation of desktop NVEs for testing video based avatars and scalability techniques. Subsequently, not much attention was given to the interactivity of the worlds therein, and its scenes consisted of statically modeled environments, apart from the user and AI controlled movable avatars.

In order to allow our approach to create more interactive scenes, we started with creating interactive object descriptions for all objects in the environment, including the avatars. Then, we implemented an interactive avatar controller handling keyboard and mouse input. This controller was used to convert the input, that was normally sent straight to application, into commands that the interactive avatar object can handle (similar to the controller example of the previous chapter). As the final step, we added the interaction layer to a basic test application and set up an *interaction server* responsible for running the simulation and distributing the client updates as was explained in the previous section 5.4. This server would run next to the standard ALVIC *game* and *world server* but on a separate machine. Although the ALVIC system does not support immersive (N)VEs, these systems can integrate our framework just as easily, by integrating it in their simulation component in the same way. The only difference will be the implementation of the controllers, as immersive systems mostly use different input devices than desktop VEs.

As already mentioned throughout this chapter, in this dissertation we do not focus on providing solutions for network distribution issues such as delay and jitter but we aim at generating more interactive and dynamic environments. The distribution architecture we utilized during testing was the CS approach outlined in Section 5.4. In order to keep the round-trip delays below the interactive threshold, the tests were held on a LAN, where delays did not exceed 100 ms. The client and server PCs were Pentium IV 1.7 GHz. computers with an internal memory of 512 MB. While we did not perform a fully formal usability test, this test environment employing our first version of the interactive object approach was utilized and tested sufficiently in order to draw some conclusions.

Regarding the interactivity, as all objects are now defined as interactive objects, the worlds have become much more lively and interactive. Avatars can trigger all kinds of behaviors by just moving around or by causing collisions with buttons or other objects. Also, objects themselves can trigger behaviors of other objects, etc. The avatar controller object work just as fine and correct as the traditional approach with no noticeable difference to the end user. Similarly, as long as the number of simultaneous scripted behaviors is not too large, their run-time interpretation does not cause extra delays or noticeable differences. Only when the number of simultaneous interactive scripts that needed interpretation at the same time grew large (over 100 interpretations) some minor delays were noticeable, specifically when the scripts started (exceeding the interactive threshold). The parsing and interpreting of course form the main problem here as they are the most resource intensive. Another partial cause of this delay can be attributed to the large number of threads that were created, as every script is run in its own thread, causing minor performance issues on the server machine that also has to perform several other thread intensive tasks. This issue could quite simply be resolved by using a more powerful machine for the server (which can not be a problem to date), since this is the only computer that needs to execute the scripts, or by keeping an interpreted version of every script at the object level, alleviating the server from the tasks of parsing and interpreting the script every time it is run. Of course the interaction layer will then need to check and compile the new scripts if behaviors are changed at run-time or when it needs to be executed for the first time. This checking is now unnecessary as each script is fully parsed and interpreted every time it is executed. Another possibility to improve script execution would be to use a more efficient scripting engine, as ours was not created specifically for performance but for usability and ease of understanding and is therefore not the most efficient. Currently several excellent commercial and non-commercial solutions exist for adding scripted behaviors to games and other real-time applications such as: Lua [Lua 07] and Python [Python 07]. Other solutions might include splitting up worlds on different servers, load balanced distributed simulation and other solutions described in the previous chapter and the literature.

With respect to network delays and the extra round-trip delay, we found that our tests on LANs always kept these beneath the interaction threshold of about 100 ms. and thus sufficed for our purposes. The network delays introduced by the deployed client/server architecture are practically unnoticeable. This is of course the combined result of the efficiency of UDP and the small packets we use for updates.

In conclusion we can state that our interactive object approach enables much more dynamic and run-time adjustable interactive virtual worlds for both VEs and NVEs. This was realized by employing a feature modeling approach coupling interaction information to the objects instead of to the application and generalizing object types. However, even though we now have created more interactive and lively scenes wherein all objects can interact and communicate with each other with a single interaction mechanism, the basic simulation approach that is lightweight and platform-independent, does not provide an entirely realistic environment. As an example consider an avatar pushing a large object such as a table on one of the corners. In our simulation approach this object will receive move messages from the avatar's object through the collision link, and perform the same movements as the avatars hands, as long as they remain in collision. As a result, the table will move forward, if the user pushes it forward. However, in a more physically realistic environment, the object would rotate as well. Figure 5.4 illustrates the difference between our basic simulation approach and the physically correct solution. Thus some further research is required to enable more realistic behavior.

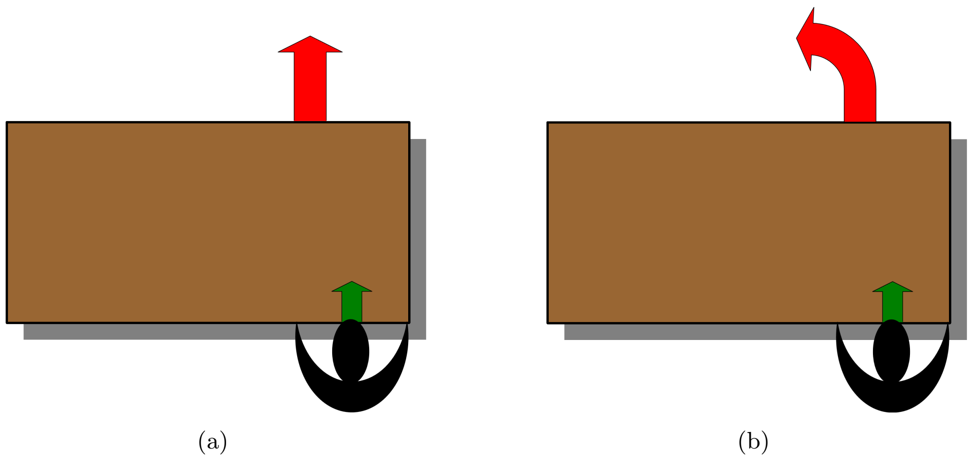


Figure 5.4: Illustration of the difference between the result of an avatar pushing on one side of a table with our basic simulation mechanism (a) and true advanced physical simulation (b). The green arrow shows the direction the avatar is pushing, the red arrow illustrates the direction the table will move in.

CHAPTER 6

Conclusions

In this part, we proposed a general framework for dynamic interactions in VEs and NVEs, that is able to provide a solution for the lack of liveliness and interactivity in contemporary systems. The presented approach employs a feature modeling approach and object type generalization in the virtual world. As a result no distinction on the interaction level is made between avatars, agents and other static or dynamic objects, and they are all subject to the same interaction mechanism. Object actions and behaviors can be performed by activating triggers or as a result of command passing through inter-object or controller interaction links. The construction of the object as well as the entire interaction paradigm are kept on the object level and objects and worlds are described in an XML based format. The interaction mechanism is implemented in an interaction layer that can easily be integrated into an application and provides the concept of controllers to control objects in the environment.

The main advantage of this approach is that all the information needed to interact with an object is located at the object level instead of at the application level, which is the standard approach. As a result, the objects, their parts, actions and behaviors can be modified, even at run-time. This could be done by the application itself, or by the application user, provided that the application developer offers support for it to the users. Also new, unknown objects can be added to the simulation at any time. While the modeling process for interactive object might require some more work than creating objects for the standard approach, these objects are highly reusable and specific interactions no longer need to be implemented in the application. Furthermore, entire objects, can be reused in other applications, and already defined parts and behaviors can be reprocessed for defining other interactive objects. Even at the network level, interactive object features could be utilized to increase performance. For example, consider an object's movement constraints. These can be checked to see if its parts can move or rotate, hereby determining if frequent updates should be sent or not. An object that has no movable parts, and that is immovable itself, should never have to be synchronized at all, since this would only consume unnecessary and scarce network resources. An object that is only able to rotate, on the other hand, should only sent updates containing its orientation.

In order to show that our solutions works in a realistic setting, our approach was integrated in the ALVIC NVE framework. Although no formal usability study was performed, several tests have shows that the integration formed no problem and that while we used only a basic distribution architecture, we managed to create more lively and dynamic VEs with only a naive lightweight simulation mechanism. We also provided some solutions for what we found to be the main performance limit, the run-time behaviors, that need to interpret and execute scripts of all interactive objects in the system.

The presented approach illustrates how we can create more dynamic and run-time adjustable virtual worlds with our dynamic object approach, however at the same time, we must admit that the realism is still not optimal. We believe that physically correct simulation and more advanced animation can increase realism even further, and consequently result in better acceptance of VEs in general. The next part will discuss how we can enable more dynamic and realistic animations and simulations and how our interactive approach can utilize these to create even more engaging virtual worlds.

Part III

CREATING MORE REALISTIC DYNAMIC ENVIRONMENTS

Table of Contents

CHAPTER 7

Introduction

In the previous section we discussed a dynamic interaction mechanism for NVEs. Although the goals of more lively and run-time adjustable virtual worlds were achieved, the simulation was rather basic, and the animations provided by the system still relied on predefined modeling, resulting in little flexibility.

In contrast to passing commands, executing behaviors and distributing virtual world states, animation is not an essential enabler of interaction for NVEs. However, proper use of animation can drastically increase the user's feeling of presence while interacting in these worlds [Casanueva 01]. This is especially true for the animation of avatars, since they are the user's means of interaction in the VE as we discussed in Chapter 2. It is through this avatar that a user can for example pick up objects, point at locations and display his actions to others in the shared virtual world. In order to achieve even the slightest level of realism, such actions clearly have to be accompanied by appropriate animations. Since NVEs are real-time applications, we can not use off-line techniques to take care of the animation in these virtual worlds as is done in the movie industry. Fortunately, real-time character animation and physical simulation have improved considerably over the past few years. Due to recent advances in computer hardware, it is currently possible to produce results in real-time that were formerly only achievable in off-line animation [Anderson 01]. In the context of NVEs however, animation has received relatively little attention. This is manifested in the fact that advanced animation techniques are only slowly finding their way into these systems. A possible explanation of this could be the computational complexity of these more advanced techniques. Another reason might be the inherent networking aspect of NVEs. Since avatars are a part of the shared world, their state has to be distributed to other connected users, and more advanced animation techniques often require more synchronization and will as a result consume more of the scarce network resources.

Fortunately, in the gaming, movie and computer graphics communities 3D computer animation has been an important research topic in the last decades. However, despite these efforts, still several problem areas exist and the demand for more realistic animations is still growing. As a result, a lot of research is focusing on improving very specific areas such as realistic human motion simulation, cloth and hair rendering, physical simulations and so on. Simultaneously, a lot of new gaming devices have entered the market over the last decades. Examples include Microsoft Xbox 360™, Sony Playstation® 3, Nintendo Wii™ and so on. These systems are, just like PCs, equipped with special 3D hardware and enough memory and processing power to show high resolution interactive 3D graphics. Also, due to recent advances in processing power and memory capacity, small portable or handheld devices such as PDAs and smart phones are currently also capable of supporting graphical user interfaces with audio and video playback. Some of these have already been equipped with special 3D hardware as well, making them suitable for more interactive 3D animated applications and games. So, it is clear that the number of platforms capable of displaying interactive 3D graphics is growing rapidly. As these portable systems are also equipped with wireless networking, many gaming and more specifically MMORPG developers are seeking ways of creating mobile clients for their systems as well. In a more serious application, a successful demonstration of enabling the ALVIC NVE system presented in [Quax 03] on a PDA are presented in [Jehaes 05].

Currently, most games and software components have been, and are still being developed for specific platforms. Since development time for games is a critical factor, not much attention (or money) is usually spent on portability or extensibility of game engines or their parts. As a result, many platform specific SDKs, libraries, game and animation engines have been developed over the years. Some specific applications or games are being ported to other platforms such as PDAs or smart phones, but since little or no consideration for these platforms was given during the design phase, the porting process is usually difficult and time-consuming. However, since more and more game developers are creating games for more than one platform, there is a trend toward more multiple platform or *platform-independent* components. In research and other less commercial areas some open platform-independent animation libraries and SDKs have been studied and developed. However, these systems are usually limited to desktop platforms (MacOS, Windows and Linux PCs). Other platforms are usually considered to be too specific or non-relevant by the developers. On the other hand, most libraries or engines for 3D games or animation are not open source or extensible in any way. Consequently, application developers can only work with the provided functionality. Some open source projects do exist, but adding new animation techniques is often very difficult or unsupported.

In this part, we elaborate on how our interactive object platform can be extended to support more realistic animation and simulation. We start by describing the *ExtReAM* (Extensible Real-time Animations for Multiple Platforms) animation library, we developed for this purpose. This library takes into account the evolution of graphics capable devices and the principles of dynamics we also used in our interaction platform. In contrast to similar systems, the *ExtReAM* library was developed to be an animation library that can easily be extended with new animation techniques and is easy to integrate in all kinds of applications, on all kinds of platforms. This is realized by implementing a very lightweight, platform-independent core system. Animation and simulation techniques are added through plug-ins that can be used by the system. Plug-ins can easily be created by developers due to the object oriented design and different platforms can use different plug-ins for similar tasks. After elaborating on this powerful animation library, we discuss how its physical simulation plug-in improves the realism of our interactive object approach and we show how our interactive object system is adjusted in order to support it. We conclude this part by discussing our findings regarding the approach.

CHAPTER 8

Related Work

8.1 Character Animation

The realistic animation of 3D objects has been widely studied, and as a result many techniques have been proposed. In 1989 Chadwick et al. [Chadwick 89] presented an efficient *layered modeling* approach decomposing a human body into distinct layers. The resulting models consist of separate discrete layers, every one of them with their separate physical and geometric properties. After the appropriate constraints are enforced between the different layers, animating the model comes down to controlling the undermost layer of the model [Giang 00]. This approach is best known as *layered modeling*. Contemporary real-time applications generally use a two layered approach consisting of a skeleton (a hierarchic structure of joints) and skin layer (a vertex mesh representing the shape). As in this case the skeleton is the layer that is to be transformed for creating animations, this approach is often referred to as *skeleton animation*. An example of a skeleton based model is shown in Figure 8.1. Due to its efficient nature and ease of use, it is still the standard animation technique in most real-time systems.

The best known and most used techniques for transforming these hierar-

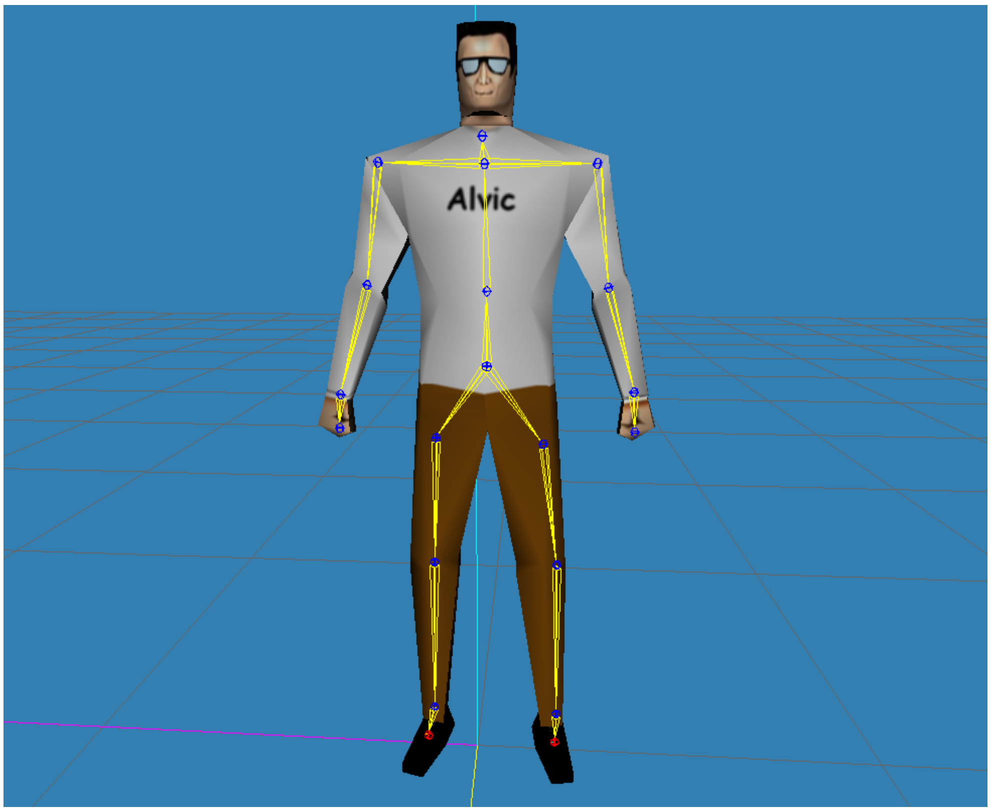


Figure 8.1: A model of a male character used in the ALVIC framework, consisting of two layers: a skeleton and a textured skin mesh.

chic 3D characters are *kinematics* and *dynamics*. Kinematic techniques are concerned with explicitly transforming the different parts of the animated objects. Important kinematic techniques include *keyframe animation* and *inverse kinematics* [Welman 89a, Giang 00].

8.1.1 Keyframe Animation

Keyframe animation consists of specifying skeleton poses (key frames) at certain points in time and then use an interpolation technique between those key frames in order to animate the objects. Key frames are mostly modeled by hand in an animation modeling application. Over the years, many systems have been developed for modeling these kinds of animations. The best known are the commercial modeling packages such as AutoDesk© 3D Studio

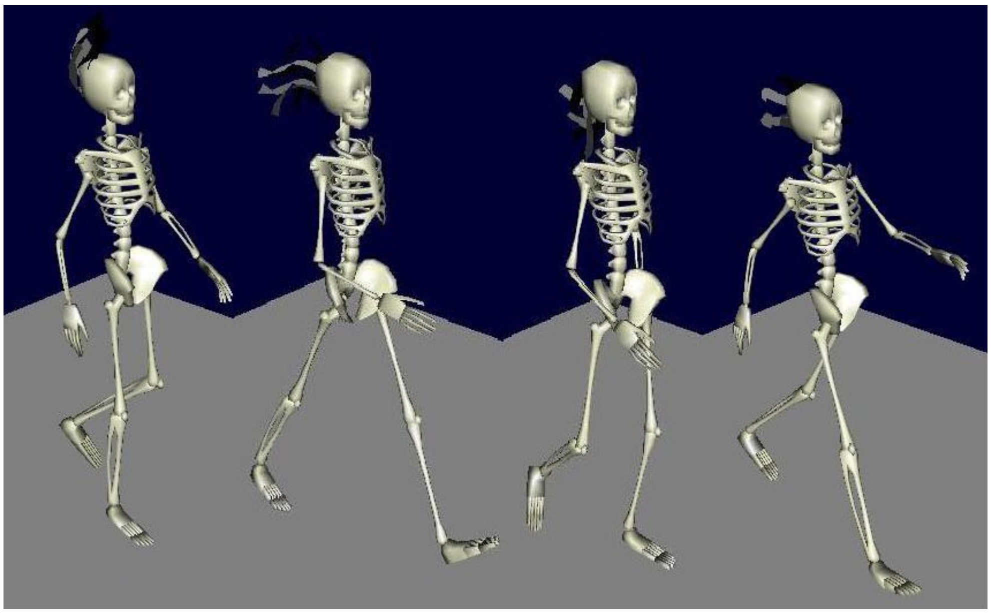


Figure 8.2: Four keyframes of a walk animation for a 3D animated model (from [Cal3D 07]).

Max [3ds Max 07], Maya [Maya 07] and Blender [Blender 07] an open source project. The generated models can be integrated into applications and the predefined animation data can be used to animate the objects, but animations can not be changed unless the application has its own animation control. It is however possible to blend different animations e.g. for different parts of an object or changing the speed of animations. Another possible way of defining key frames is through motion capture which can also be used to control an animated figure directly. Motion graphs [Kovar 02] on the other hand use a database of kinematic motions to automatically calculate transitions between key frames. Keyframe animation is an example of a *forward kinematics* animation technique, since it specifies all joint transformations in order to have the end of the chains reach a certain position. A sequence of four frames taken from a walk cycle of an animated 3D character is shown in Figure 8.2.

8.1.2 Inverse Kinematics

Inverse Kinematics (IK) techniques, on the other hand, originate from the field of robotics. They can be used to dynamically generate motions for (part of)

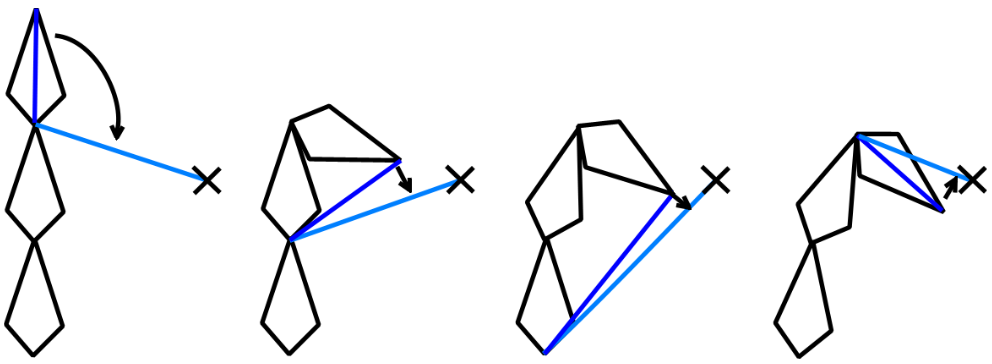


Figure 8.3: Illustrations of a numerical IK method in 2D taking discrete steps to move the IK chain's end effector closer to the goal position.

a skeleton. Normally a chain of bones with a specified end-effector (the final joint in that chain), is instructed to move toward a specified goal position. This move toward that goal position is automatically calculated [Welman 89b]. A possible example of an IK chain could be the arm of a virtual human, with the hand being the end-effector. IK solutions can nowadays easily be calculated on-the-fly, resulting in real-time adjusted animations [Anderson 01]. However, the complexity (and as a result calculation times) increases with the addition of every bone and constraint. Broadly speaking, inverse kinematics algorithms can be characterized as *analytical* or *numerical*. Analytical methods attempt to mathematically solve an exact solution by directly inverting the forward kinematics equations. This is only possible on relatively simple chains with limited DOF. In contrast, numerical methods use approximation and iteration to converge toward a solution as illustrated in Figure 8.3. They tend to be more expensive, but far more general purpose.

8.1.3 Dynamic Animation Techniques

Dynamic, in contrast to kinematic, animation techniques use physical forces and laws to simulate object movements. Since creating physical controllers for complex articulated 3D characters is not a trivial task [Faloutsos 01a], dynamic animation techniques in games are usually limited to ragdoll physics[1], which simulates lifeless bodies. For example, when in a first-person shooter somebody is shot, the user looses control over his character, and the physics engine takes over, creating a realistic body response. An example of the results

[1]Wikipedia - Ragdoll Physics (`http://en.wikipedia.org/wiki/Ragdoll_physics`)

of ragdoll physics is shown in Figure 8.4. Hybrid techniques have also been investigated in the past [Shapiro 03, Zordan 02], and are still being studied to date. The main goal usually includes achieving more control over dynamically animated characters. However, attempts to adjust kinematic motions with dynamic effects to improve realism and variation have also been explored.

Endorphin [Endorphin 07] is commercial modeling package supporting dynamic animation techniques. It uses *dynamic motion synthesis* and adaptive behaviors to create very realistic animated 3D characters. Dance (Dynamic Animation and Control Environment) [Faloutsos 01b], on the other hand, is an open and extensible framework for computer animation focused on the development of physically based controllers for articulated figures. These systems provide the tools to create physically simulated animations but are not suited to be integrated into other applications.

In contrast to animation modeling applications, animation libraries are specifically designed to be easily incorporated into other software. Cal3d [Cal3D 07], for example, is a skeleton based 3D character animation library written in C++ in a platform- and graphics API-independent way. It can easily be integrated into different applications and provides basic skeleton animation techniques such as forward kinematics, keyframe animations and animation blending. However, Cal3D is limited to these animation techniques since it was specifically designed for only these tasks and provides little means to extend its functionality. Granny 3D [Granny3D 07] is another commercial animation system that can easily be integrated into applications. It has powerful support for all kinds of skeleton animations and is available for several platforms. Extensibility is however hardly provided.

8.2 Rigid Body Simulation in Real-time Virtual Environments

Physically based simulation techniques are already widespread in off-line animations such as in the movie industry, but with the computational capacity of modern computers, these techniques are also slowly finding their way into real-time computer applications. A relatively simple, and often employed technology is *rigid body simulation*, where the position and orientation of the objects are calculated using simple laws of physics. Many middle-ware products exist that can calculate *rigid body dynamics* efficiently, and we discuss some of the more important ones here.

Havok [Havok 07] is a commercial solution that has been proven to be

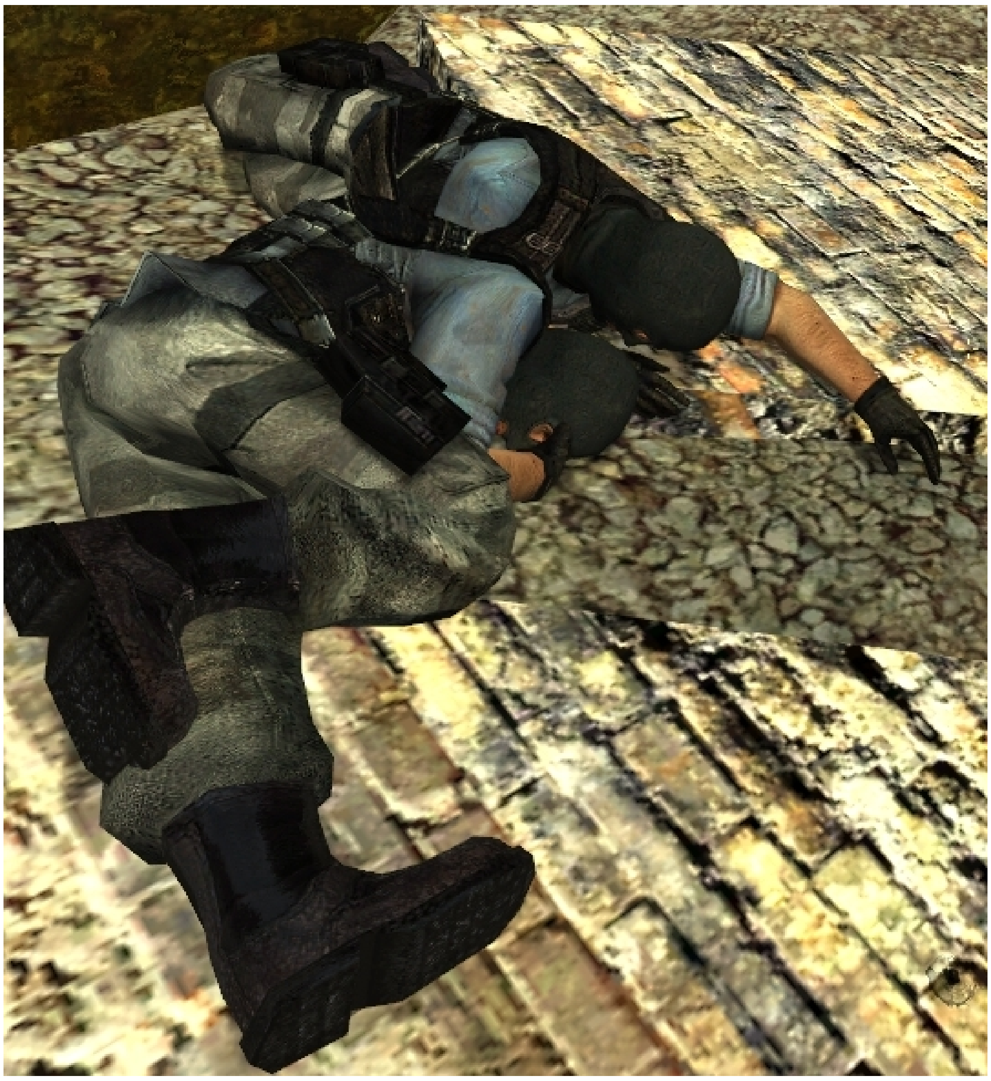


Figure 8.4: Ragdolls of two virtual terrorrist's avatars. After being shot, the physics engine took over to create realistic movement (from Counter-Strike^{TM}).

one of the fastest and most stable solutions available. It has won several awards and is used in commercial games, like *Max Payne® 2* [Payne 07] and *Half Life® 2* [Half-life 2 07]. *Ageia PhysX*™ [PhysX 07] is another commercial product, but is free for non-commercial use. It has a complex test-suite available to tweak the engine and make it even more stable. It is employed in for example the *Unreal® 3* [Unreal 3 07] engine. *Open Dynamics Engine*™ *(ODE)* [ODE 07] is probably the most used open-source solution for *rigid body simulation.* Recent releases added a new mathematical solver, making it stable and fast enough to be used in commercial applications. *Newton Game Dynamics* [Dynamics 07] is a recent open-source solution, that is rapidly growing to be a successful competitor to *ODE.*

As already mentioned, these tools are rapidly becoming very popular in the gaming industry. Although very simple physically based techniques were used before as eye-candy (mostly particle systems with gravity and collision detection), games such as *Half-Life® 2* [Half-life 2 07] use the middle-ware products mentioned above to make dynamic objects part of the gameplay (see Figure 2.4). Thus apart from making virtual worlds look more realistic, rigid bodies can also largely influence interaction. For example while in the standard approach, a button, such as a light switch, can only be pushed by an actor, in the physical approach, the collision and force information can be used. As a result the light could also be switched on or off when a non-actor object collides with it. Looking at more professional NVE applications, we see that physical simulation, if present at all, is usually limited to some form of gravity and collision detection on the avatars and the world itself. Most objects in the world would be on a fixed position in the world, not even movable and mostly not interactive at all.

The following chapter describes the animation library we developed in order to allow multiple and new animation techniques to be applicable in real-time NVEs in order to achieve a higher level of realism. Furthermore, we present how this system employs plug-ins in order to allow the system to be extended with all kinds of animation techniques and file formats without changing the core.

CHAPTER 9

The ExtReAM Library: Extensible Real-time Animations for Multiple Platforms

9.1 System Overview

In order to be as lightweight and portable as possible, the *ExtReAM* library is built around a very lightweight core. This core is primarily responsible for object management, registering plug-ins and plug-in components. Furthermore, it has a built-in event and command system that can be used by the plug-ins. In this way, plug-ins can register functions that can then be executed by an application or other plug-ins in a similar way as interactive objects can call commands. As an example consider a keyframe animation plug-in, registering functionality to start and stop an animation in order to allow the application that loaded the plug-in, or another loaded plug-in to start and stop animations of a specific object instance. The more platform-independent and resource consuming tasks such as file loading, object creation, command handling and the animation techniques are left to be implemented in the plug-ins. This results in a very flexible system that can be be integrated easily in all kinds of applications on many different platforms. Figure 9.1 shows how

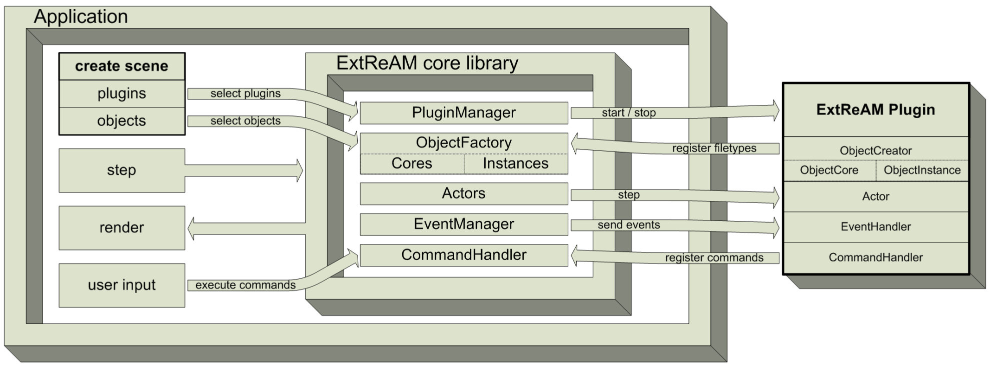


Figure 9.1: Overview of the *ExtReAM* library in an application.

the *ExtReAM* library is used in an application. The application chooses which plug-ins and objects to load. A Plug-in can be loaded into memory for just as long as its functionality is required. As a result, it will only consume resources when necessary. For example plug-ins for loading certain object types will only be necessary when the scene is created. Thereafter, the plug-in can be released, freeing resources.

Because the *ExtReAM* system is only an animation library and not a graphics engine, a rendering component is not included. All the necessary data for rendering (vertex buffers, textures,...) can however trivially be retrieved from the system. In order to advance animations and simulations, the application can step the library at every time frame. Commands can be executed by communicating them to the *command handler*, which is part of the core. Once an actual plug-in is loaded, interaction with it is completely transparent from the application's perspective. Plug-ins can also register their own commands, enabling the application to use their functionality.

9.1.1 Plug-in System

Plug-ins are used extensively in the *ExtReAM* library. A plug-in is a "building block" on top of the core system. Plug-ins can perform common tasks like loading mesh files or specific tasks like animating a skeleton-based character. As plug-ins are one of the main components of the system, providing the actual functionality, we assured that creating a new plug-in is as simple as possible. As a result of the object-oriented design, in order to create a new plug-in, the developer must only derive from a (couple of) class(es) and implement a few methods. More specifically, for most of the plug-ins, it involves deriving from the `Plug-in` class and implementing the `start` and `stop` methods.

The core system internally uses a *plug-in manager* to start and stop plug-ins and to get information from certain plug-ins. It also ensures that plug-in dependencies (other plug-ins that it relies on) are started first and that plug-ins are not loaded more than once.

9.1.2 Plug-in Components

To extend the core animation system, using the principles of object-oriented design, the *ExtReAM* library provides some abstract base classes that can be used in plug-ins in a standard object-oriented way. Here we give an overview of the different base classes and explain how to use them.

To provide a new *object type* in the *ExtReAM* system, three classes have to be implemented. Examples of object types might include: skeleton objects, rigid and soft body objects, ... How the actual objects are managed in the system is explained in section 9.1.3.

- `ObjectCore`: an object core is generated from the data that gets read from file. From this core data, one or more *object instances* can be created. The instances can share the core data if desired, so redundant data can be minimized.

- `ObjectInstance`: an object instance is an actual entity in the virtual world. It is created from an *object core* and can use that core's data besides having its own specific data.

- `ObjectCreator`: an object creator can register itself in the *object factory* (a part of the core) with certain file types, and is used to create object cores from file and object instances from these cores.

The remainder of the plug-in base classes are used to alter the behavior of the animation system:

- `Actor`: actors are used to alter the scene every timestep. The elapsed time is provided every frame when the `step` method is called. We have used actors, for example, to advance the physics system and blend skeleton-based animations (see sections 9.2.1 and 9.2.2).

- `EventHandler`: this class has to be implemented if a plug-in needs to react on certain events in the system, like objects being added or removed. An example is our *rigid body simulation* plug-in (see section 9.2.1), where a physical shape is automatically created when an object instance is added, and deleted when the object instance is removed from the system.

- `CommandHandler`: through this class, a plug-in can register the commands it can handle. By using commands, the coupling with plug-ins and the actual application can be very loose (the application does not have to know anything about the plug-ins). The core system has built-in commands to start or stop plug-ins, add or remove objects and alter their position or orientation. Examples of registered commands can be found in the *KeyFrameController* plug-in (see section 9.2.2), where we register commands to start, stop and pause keyframe animations for skeleton object instances.

9.1.3 Object Management

The *ExtReAM* system is designed to easily manage object creation and removal in the scene, as efficiently as possible. The library keeps track of *object core*s and *object instance*s. The object core contains all necessary data to create one or more object instances, the actual entities that will be rendered in a scene. *Rigid objects* are a good example: the core contains the geometry to be rendered and the bounding geometry to be used with collision detection. All that the actual instances need is a position and an orientation in the virtual world.

Creation of object cores and instances in the system is handled by the *Object Factory*. The factory automatically chooses the right creator if the user asks to create an object. The actual creators from which the factory can choose can be implemented as plug-ins as explained in the previous section. The system can then select the correct creator plug-in, based on the registration of the object types they can create, and the file types they can read. Furthermore, the factory makes sure a file is not read twice if that is unnecessary.

9.2 Plug-in Examples

9.2.1 Rigid Objects, Bodies and Physical Simulation

As the main goal of the animation library is to add more realism to our virtual worlds, we started with the creation of a *rigid body simulation* system. In order to achieve the most flexible solution, the implementation is split up into two plug-ins.

Firstly, we created the `RigidObject` plug-in, which is able to create a basic physical object type: `RigidObject`. The RigidObject's core is able to store an object's geometry and bounding box. RigidObject instances on the other hand, can have a position and an orientation. In order to be able to create these objects from different file types, we also constructed several *object creator* plug-ins that each can create an object from one or more different file formats. Some of the file types supported include ply, 3ds, Ogre, ms3d, etc.

Secondly, we created the actual `Physics` plug-in, that encapsulates the *Ageia PhysX* [PhysX 07] engine and provides the rigid body simulation control. The plug-in contains an *actor* that advances the physical scene every timestep through the inherited `step` function. Furthermore, it implements some physics specific functionality, such as creating physical bodies from triangle meshes. Unfortunately, the *Ageia PhysX* engine is currently only available for desktop computers and some consoles. It is unavailable for mobile or handheld devices. As a result, this `Physics` plug-in can not be used on these platforms. This is a perfect example of a plug-ins that could be implemented differently for different platforms. We could for example use the simplified simulation approach described in the previous part, to create a more lightweight and less memory consuming plug-in to be used on less powerful devices. The `Physics` plug-in automatically couples physical shapes to rigid objects added to the scene to be used for collision detection and rigid body dynamics. Logically, the physics plug-in is dependent on the `RigidObject` plug-in.

After every time step, the actual positions and orientations of objects are set according to the physical shapes that are simulated, resulting in physically correct motions.

9.2.2 Skeleton Animation System

Since skeleton animation is one of the most applied character animation techniques, we also created a *skeleton animation system* for the *ExtReAM* library. Similarly as the physics plug-ins, in order to improve flexibility, we provided this through a set of different plug-ins.

The `SkeletonObject` type has an object core containing the geometry and a core skeleton structure. Also, it contains the link constraints between the skeleton and the geometry to do *vertex blending* [Lander 98]. The `SkeletonObject` instances on their part, have a position and an orientation in the virtual world, and also contain their own deformed skeleton pose and if desired a deformed geometry. Both are implemented in the `SkeletonObject` plug-in.

The bone structure we use is the standard tree structure used for skeleton-based animation: it consists of a single root bone and every other bone has a parent and a list of children. Furthermore, every bone stores a position and an orientation relative to its parent. Different *skeleton controllers* (which can be implemented as different plug-ins) can animate the skeleton. Every controller can create a skeleton pose for every instance, and a *skeleton blender* combines the poses of all the controllers into the final pose for the skeleton object instance. This blender can use interpolation between the final skeleton poses of the different active controllers or select one as the most important. This can be controlled by the application or by another plug-in. The skeleton blender can also apply weights to the different controllers so that one controller can have more influence on the entire skeleton or a specific part of it. An example of when this might be used is when we manipulate an avatar's hand for grasping something using IK while it is performing a keyframe 'walk' animation.

To deform the geometry, a *mesh deformer* was implemented. This deformer uses the final *skeleton pose* generated by the *skeleton blender* to generate a deformed skin. It uses the standard weighted vertex blending technique [Lander 98]. If the deformed geometry is not needed in software, a graphics hardware method could be used to speed up the deformation. Our current system only provides a software implementation of vertex blending, so the same technique can be used on all platforms. Separate plug-ins could be created for the different platforms, wherein more platform-specific and performant techniques could be used. On desktop platforms for example, the vertex blending could be implemented using *GPU* hardware. These skeleton animation specific components are grouped in the `SkeletonObject` plug-in. Figure 9.2 shows an overview of the skeleton animation system.

In order to provide some basic animation techniques, several skeleton controllers were already implemented in this system which we will briefly be discussed now.

The KeyFrame Controller

To support standard keyframe skeleton animations, the first skeleton controller we propose is the *keyframe controller.* For each `SkeletonObject` core, it holds a list of available animations, that can be read by the skeleton object creators. An animation consists of a list of *key frames* describing a (partial) skeleton pose at a certain time. In this way, animations can be defined on an entire or a specific part of the skeleton. When an animation is started for a certain object instance, the plug-in creates a new *animation state* that holds the current time

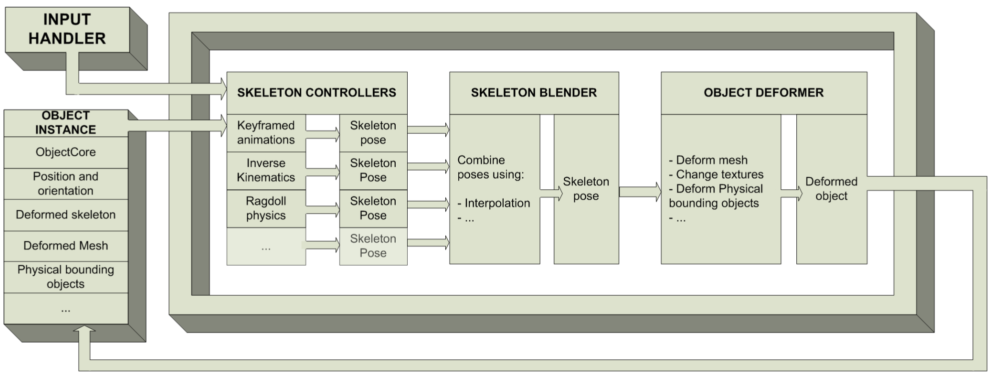


Figure 9.2: Overview of the skeleton animation plug-in.

and the weight of that specific keyframe animation. The keyframe controller then calculates a skeleton pose by interpolating between the keyframes of the running animations. Finally, a *keyframe command handler* is created, implementing functionality to start and stop animations and apply weights to them.

The IK Controller

In order to allow on-the-fly motions to be created as well, the second motion controller allows (parts of) poses to be calculated using an IK technique. The current plug-in implements the Cyclic Coordinate Descent (CCD) technique [Welman 89a]. This is, together with the Jacobian Transpose (JT) method one of the most applied IK algorithms to compute real-time IK solutions for animated skeletons. Although each CCD iteration is somewhat more expensive to calculate in comparison to JT, it generally tends to require far fewer iterations to converge to an acceptable solution. In order to calculate a pose, the algorithm takes a part, or the entire skeleton of an animated character, an end-effector and a goal position. Joints can also specify constraints, in order to avoid unrealistic or impossible poses. Furthermore, as CCD tends to favor joints closer to the end, we also allow joints to define a damping factor, which limits the maximum number of degrees a joint can be rotated per iteration. Finally, since we are working in real-time, the IK calculation request can specify a maximum time for calculation. In this way, the total amount of time for iterating can be limited, resulting in the best possible solution given the time limit. After calculation, a skeleton pose for the IK chain will be provided to the skeleton blender.

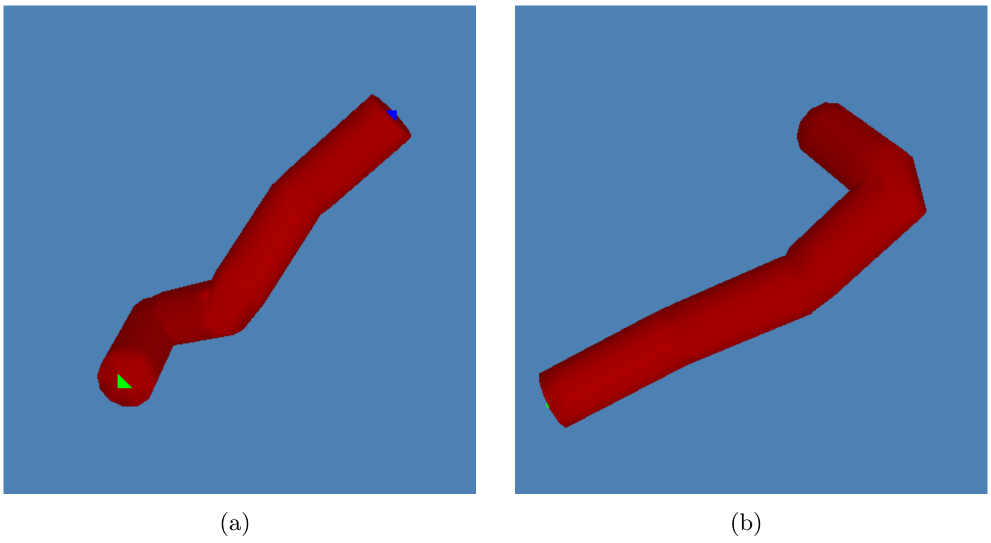


Figure 9.3: Two different IK results for the same IK chain. (a) Has constraints and strong damping for the last two joints in the chain. (b) Constrains only the second joint and has no damping on the final joint.

Physical Controllers

In order to demonstrate that our system can also provide dynamic animation techniques, two physical controllers were also implemented. The first one is the `RagdollController` plug-in which employs the `Physics` plug-in which we described in Section 9.2.1 to implement a ragdoll system. This plug-in automatically builds an articulated rigid body structure from a skeleton object instance for the physics engine and submits it to the rigid body simulation system. After every timestep, a skeleton pose is calculated from the movement of the physical objects.

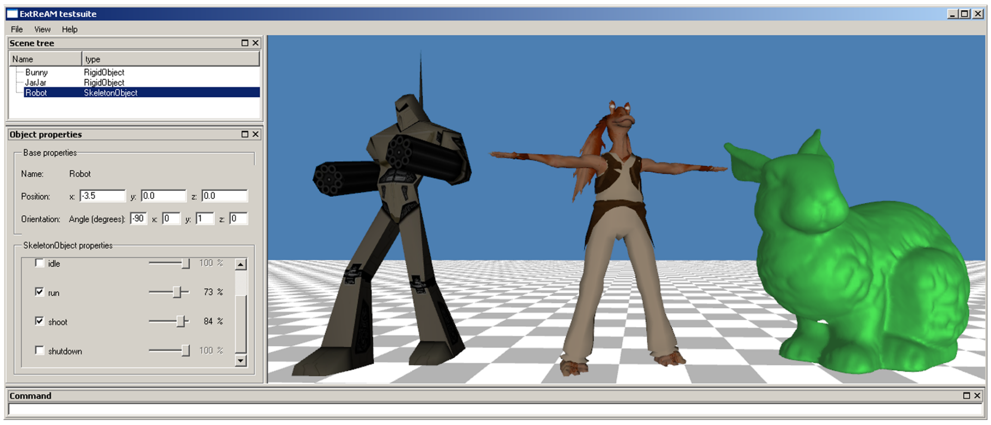
A second more advanced physical controller is an extension to this ragdoll system and implements a simple dynamic controller enabling an animated object to stand upright and stay balanced. This `BalanceController` was created specifically for human models and uses joint motors on the ankles to keep the model in balance. This implementation only contains a basic solution, which will not be able to keep the model balanced in all circumstances. More advanced balance strategies, such as swinging the arms or performing protective steps could be added as well. However, as dynamic animation is a subject on its own, especially when it comes to animating humans in a realistic physical way, we will not discuss this subject any further.

9.3 Evaluation of the ExtReAM library

9.3.1 Desktop Evaluation

Before we integrated *ExtReAM* into our interactive object approach, we first wanted to check its portability and functionality. Therefore, a test application was developed for desktop platforms and was tested under MS Windows and Linux. The application provides functionality to load scenes described in a simple *XML*-based scene format that contains the necessary plug-ins and object positioning. Furthermore, the user can load extra plug-ins and add, remove and reposition objects at runtime. The interface was developed with the *QT 4.0* library [Qt 07] and shows a tree structure of the objects and object specific properties. These properties are specific for every *object type*. For example, a skeleton object instance has a property displaying the possible keyframe animations that it supports and the weights that are applied to them. The system also supports command handling, so users can send commands to the system for example to invoke core or plug-in functionality. OpenGL is used to render the scenes.

All the controller plug-ins described in the previous sections were tested. Integrating the library and using plug-ins was straightforward, and the plug-in system did not introduce any noticeable slowdowns. Figure 9.4 shows a screenshot of the test application containing a scene with one skeleton-based object and two rigid object models. The user is controlling the keyframed animations of the skeleton object using the properties widget for skeleton objects. We did not optimize the rendering, since this was not part of our research. However, even without this optimization, the system can render a sufficiently large number of objects for our purposes. On a standard PC, Pentium 4, 2.4 GHz with 1 GB RAM and an ATI Radeaon 9800 graphics card and no physics processor, we managed to animate and render over 250 keyframe animated objects (a model with 18 animated joints and 400 mesh triangles) and approximately 900 simulated movable rigid body objects (non-convex, with an average of 48 triangles per object). With more extensive use of rendering hardware, a physics processor [PhysX 07] and optimizations of the physically simulated objects (using convex objects, or basic physics shapes) these results could be drastically improved.



Name	type
Bunny	RigidObject
JarJar	RigidObject
Robot	SkeletonObject

Base properties

Name: Robot

Position: x: -3.5 y: 0.0 z: 0.0

Orientation: Angle (degrees): -90 x: 0 y: 1 z: 0

SkeletonObject properties

☐	idle	100 %
✓	run	73 %
✓	shoot	84 %
☐	shutdown	100 %

Figure 9.4: The desktop test application, using QT and OpenGL. The skeleton object properties widget (bottom left) allows the user to activate animations and set weight factors.

9.3.2 PocketPC Evaluation

As a second test, aimed at portability, we developed a PocketPC application using native C++, and *OpenGL ES* as the rendering backend. This was tested on Dell Axim X50v PDAs, that are equipped with hardware-accelerated 3D graphics, so we could test advanced animation techniques without the delays caused by software rendering. The PocketPC application can load the same scenes as the desktop application, but can't load the desktop-specific plug-ins such as the `Physics` plug-in we presented in section 9.2 as it relies on the Ageia PhysX library which is only available for desktop platforms

We found that the plug-in-based system is an efficient solution for these kind of devices which have only limited memory resources, because plug-ins (along with their linked libraries and memory usage) can be unloaded when their functionality is no longer required. The *object creator* plug-ins for example, that are used to generate object cores and instances from file, are loaded only when necessary, and are immediately unloaded after the objects have been created.

Figure 9.5 shows a scene consisting of 18 *rigid objects* (each about 1800 polygons) and one animated *skeleton object* (with 400 polygons and 18 animated skeleton joints). The scene is rendered in real-time (20 frames per second). The plug-ins used are the same as in the desktop application. *Vertex blending* forms the biggest bottleneck for this scene, since the algorithm is completely running in software and thus not optimized for the device.

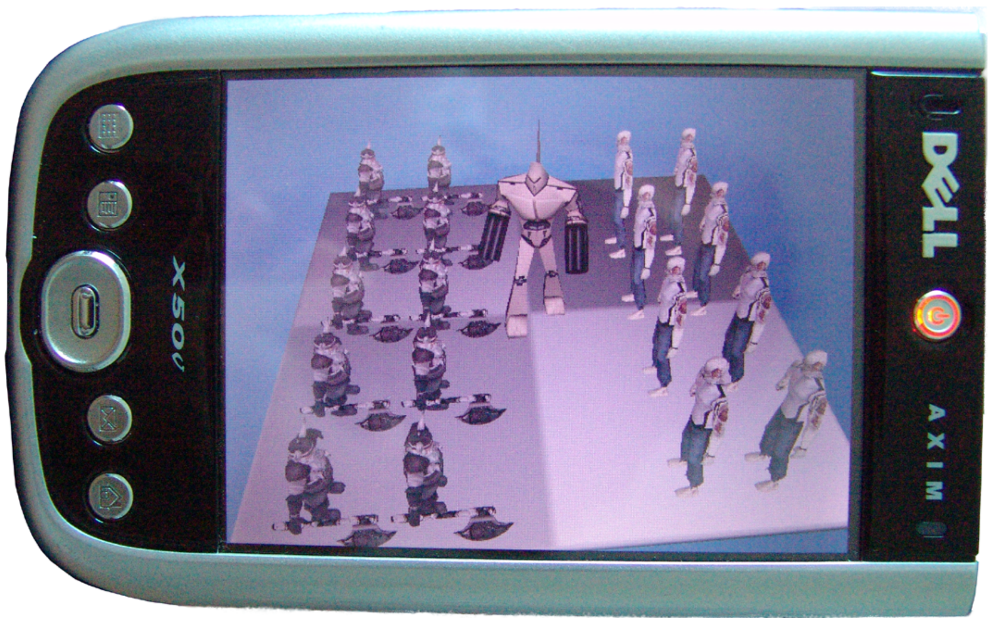


Figure 9.5: PocketPC application. The robot is a *skeleton object* performing the “walk” animation. The other objects are *rigid.*

CHAPTER 10

A Physically Simulated Dynamic Interactive Object System

In the previous chapter we elaborated on the *ExtReAM* library, supporting realistic, dynamic animation and simulation techniques. In this chapter we will discuss how it is integrated in our interactive object system in order to provide applications with this functionality as well.

In order for the integration to be successful, several adjustments need to be made to the original system which we described in Chapter 5. First of all, the animation library must be integrated in the interaction layer, so realistic animation, and physical simulations can become an intrinsic part of the interaction system. Secondly, if we want objects and worlds to behave more natural, their descriptions need to be extended, allowing them to describe properties and actions related to physical simulation as well. Finally, in order to create dynamic reactions, object behaviors should also be able to express physical actions and offer support for the new animation techniques to be controlled as well. Finally, as realism is not elementary for all kinds of VE systems and to remain backwards compatible, we want these parts to be optional, so the basic version of simulation must also be supported.

10.1 Improving Realism in the Interaction Platform

As we explained in Chapter 5, the *Interaction Layer* is the component responsible for handling the interactions and the simulation of the virtual worlds. In the first version, a basic simulation system was provided, that supported collision detection and movements in a basic way. In order to keep support for this lightweight approach, this component is separated from the core of the interaction system into a separate library. This allows an application to specify whether it uses the basic rather then the *ExtReAM* system, without requiring both components to be loaded. Thus when the basic simulation component is selected, the simulation system works as explained in the previous Part. However, when an application chooses *ExtReAM* to handle the simulations and animations, the interactive object system will enable a whole new range of realism. First of all, the simulation can be done by loading and employing the `Physics` plug-in that we presented in the section 9.2, enabling rigid body dynamics.

In order to make sure that the interactions themselves or behaviors of objects do not change, the communication mechanism, employing interaction links and command messages, is left unaltered. Thus, the objects will still receive movement commands of connected objects for example. However, the rigid body simulation mechanism ignores these move messages and calculates all movements and collision responses on its own. This results in a much more realistic experience since it makes results of interactions look more physically correct. The only extra that needed to be implemented in the simulation system was the possibility to allow the interaction layer to request object collisions, a task that in the basic simulation system is provided by the collision manager. However, the integrated physics engine supports this in a similar way.

Apart from simulation, using the new library also provides a number of new animation techniques that can be used when the correct plug-ins are loaded. For example the skeleton animation plug-ins, allows keyframe, IK and even dynamic motions to be created on-the-fly. Of course in order to support all this new functionality, new commands must be provided. Therefore, we allow the interaction system to reuse and forward commands that are registered in the *ExtReAM* plug-ins. In this way, we can instantly use all functionality from these plug-ins for defining actions and behaviors for our interactive objects. For the behaviors to be able to call these functions, the scripting engine was also adjusted in order to support *ExtReAM* commands. Of course when such a command is requested, the application developer must ensure that the correct plug-ins are loaded or the command will fail. In case e.g. the IK plug-in is not loaded and an action `moveIKChain` is requested, the action will not be executed. Also, object controllers can utilize the animation plug-ins' commands by forwarding them to the animation library. In this way, application developers can directly control object animations and simulation.

In order to be able to use objects from the basic approach, we implemented a conversion for the `move` and `rotate` actions that formerly used no physics, to new physical functionalities that use forces in order to generate the move. These forces are calculated automatically based on the duration of the action and the mass of the object that needs to be moved. The scripting engine is finally also allowed to change physics parameters, so these can be adjusted at run-time as well, similarly as the other properties of the interactive objects.

In order to define the properties necessary for the new system requires some changes to the object and world definitions are required. These extensions are the subject of the next section.

10.2 Extending Object and World Descriptions

To determine if worlds are physically simulated or using the basic approach, the interactive world description is adapted to allow this to be specified. Furthermore, when physical simulation is enabled, the world designer must be able to specify some global physical properties. First of all, the force of gravity can be set. Since virtual worlds are not necessarily earth-based, any force is allowed (within the limits of the physics engine). Secondly, we allow physical joints to be defined that connect objects. In order to be independent of the physics engine, these are specified in a general way. These joints can constrain the movements of objects relative to each other. Finally, since we are working in physical worlds, we need to be able to define the materials that the objects are made of. The reason we define materials at world level, instead of at object level, is that many of the materials are reused for different objects. This way, when we want a material to change some of its parameters, it has to be done only once for all objects in the world. Of course, these can be incorporated from a file as well, so they are easy to reuse in different worlds. Two examples of how materials can be defined in the world's XML description are shown in Listing 10.1. Furthermore, this listing shows how rigid body simulation is enabled and how the gravity is set.

Listing 10.1: Part of a physical world's XML description showing the definition of materials and the global physical settings.

```
<CIWORLD worldname="testworld" rigidbodies="TRUE" ygravity="-9.81">

  <MATERIAL name="unbreakableglass" restitution="0.7"
            staticfriction="0.5" dynamicfriction="0.5" />

  <MATERIAL name="steel" restitution="0"
            staticfriction="0.5" dynamicfriction="0.5" />
```

Similarly as the world description, the object descriptions are first of all extended with a number of physical specifications. Parts are extended to describe the material out of which they are constructed. These materials can be world specified materials, or object specific. The interaction system separates these materials; object specific materials will overwrite a world material with the same identifier, but only for instances of that object type. In order to be able to overwrite the automatically calculated collision object, an object modeler can specify the collision model for a part. This is often useful, when a part is graphically complex but physically less detail is required, or when a part closely relates to a standard collision shape. Overwriting the calculated collision mesh with a simpler, less detailed mesh or a standard shape (sphere, box, cylinder, ...) can significantly increase performance. Secondly, similarly as objects can be interconnected, object part relations can also be specified with the same physical joints in the object properties. Thirdly, object actions can now also be defined as strings with a string parameter set. This allows them to define unknown commands that can be handled by e.g. plug-ins of the *ExtReAM* library. As an example consider the door object presented in Chapter 5.2. Instead of just moving the doors, actions could be defined that use physical forces and velocities to move the doors, by employing `Physics` plug-in commands. The three changes are illustrated in Listing 10.2. This listing consists of pieces of a physical version of the glass door example described in Section 5.2. First a physical part is described integrating a physical collisionbox with a material that was specified at world level (see Listing 10.1). Furthermore a prismatic joint is specified, that allows the sliding door to move 2 meters to the left. Note that this is similar to the part's constraints. The reason both are defined is that in this way the object can be used in both physical and non-physical worlds. Finally we describe an object action that utilizes a rigid body plug-in function for setting a physical object's velocity. The resulting force will close the left sliding door. The entire object description is given in Appendix C.2. In order to use the physical action in the behavior instead of the standard open door action, the script just needs to change the name of the action called.

Listing 10.2: Parts of the physical glassdoor object: a physical part, a physical joint and a physical action.

```
<!-- Physical part-->
<PART partid="glassdoor_left" filename="glassdoor.ms3d" parentid="doorframe">
    <POSITION x="-2.9" y="1.85" z="0.1" />
    <ORIENTATION x="0" y="0" z="0" />
    <PHYSICSCOLLISIONBOX
    xsize="2" ysize="3.8" zsize="0.1"
    xpos="0" ypos="0" zpos="0"
    xrot="0" yrot="0" zrot="0" materialname="unbreakableglass" />

    <PART_CONSTRAINTS>
      <MAX_TRANSLATE_CONSTRAINT upx="2.0" upy="0" upz="0"
          lowerx="0" lowery="0" lowerz="0" />
      <MAX_ROTATE_CONSTRAINT clockwisex="0" clockwisey="0" clockwisez="0"
          cclockwisex="0" cclockwisey="0" cclockwisez="0" />
    </PART_CONSTRAINTS>
  </PART>

  <!-- Physical Joint -->
  <JOINTS>
    <PRISMATICJOINT jointname="sliderleft" partid="fixedglass_left" partid2="glassdoor_left"
        xorientation="1" yorientation="0" zorientation="0"
        xleftextent="0" xrightextent="2" />
  </JOINTS>

  <!-- Physical Action -->
  <ACTION name="close_left_velocity">     <!-- Physical action -->
    <SET_PART_LINEARVELOCITY partid="glassdoor_left" xvel="5" yvel="0" zvel="0" time="2000" />
  </ACTION>
```

As the interaction properties only provide an interface and a way to describe triggers, no changes are required to allow more realism to be described. Similarly, although the scripting engine was adjusted, as scripts are described as a string in the object description, the specification of object behaviors in the file format required no changes. Table 10.2 gives an overview of what is actually stored in the extended object description.

10.3 Results

In order to test if our physical approach works just as well as the basic approach, we integrated the new version in the ALVIC framework, similarly as was done in Chapter 5.5. We used similar machines for testing the new approach in a LAN environment and used the same network setup. For the human avatars, we constructed a specific interactive object, with collision objects for the body, the arms and hand. An avatar controller was also implemented, that allows navigation and hand movements. Navigation behavior also triggers the walk animation, through the `startKeyframeAnimation` command, that is forwarded to the *ExtReAM* library. The controller also enables the user to move the avatar's hand around. This is animated by controlling the skeleton on-the-fly with the IK component of the animation library. More information on the avatar controller, the possible interaction techniques and the resulting

Object Properties	• Object specific materials • Parts, consisting of: – Relative position and orientation – 3D description – Physical description • Joints connecting the parts and constraining their moves • Basic Actions • State variables
Interaction Properties	• Commands • Interaction Zones • Triggers
Object Behaviors	1. Behaviors 2. Command to action/behavior mappings 3. Trigger to action/behavior mappings

Table 10.1: An overview of the contents of the extended object description.

animations will be discussed in the next Part.

Whereas the basic interactive object system enabled many new kinds of interaction features, especially with respect to run-time adjustments, the physical simulation component replaces basic collision detection with physically realistic reactions of all objects (apart from deformations). As a result, a user can now push objects forward, tip them over, smash them off each other and so on, making the experience much more alive, realistic and explorative. Figure 10.1 illustrates some of the results of the new interactive object system employing our animation library with rigid body simulation. On the left side, an illustration of how an avatar pushes away one of the bottom boxes of a stack, causing the whole stack to collapse is shown. The physics engine, that is now an inherent part of the interaction component, automatically calculates realistic motions. The right side shows how interaction and simulation are now trivially combined in the new interaction platform. The avatar has pushed the first of a row of dominoes, causing the rest to tumble as well. When the last block of the row falls, it collides with the button, which is part of a collision trigger of the door object we described in Chapter 5.5. This causes the `MoveDoorScript` behavior to be triggered, closing the door. Only this time, the `closeDoor` actions use physical forces to move the door's panels. As a result, the door causes a second row of dominoes to tumble. This is also a perfect example of an inter-object interaction that would in a traditional NVE interaction approach be unavailable, since they program these kinds of interactions specifically for each case, and often only avatar-object interactions are provided. Using our approach, these new kinds of interactions are trivially supported, creating more interactive and dynamic virtual worlds with minimal efforts.

Finally, we mention that this physical interactive object approach can also have an impact on collaboration and can enable new cooperative modes of interaction. E.g. since objects have physical shapes and masses, and avatars can be used to apply forces on objects, it is possible to create situations in which users have to work together in order to achieve a common goal. Consider a large and heavy object blocking an entrance, one avatar could not have enough force to move it on its own. It would require the force of several avatars, and thus the cooperation between different users or agents, to move the object away. These kinds of cooperations can be necessary in training tasks for example. Of course, the object's physical models and avatar controllers would need to be attuned to each other. However, this is not an unsurmountable problem if the same basis for mass and force are used, for example real-world values, which can perfectly be handled by most physics engines.

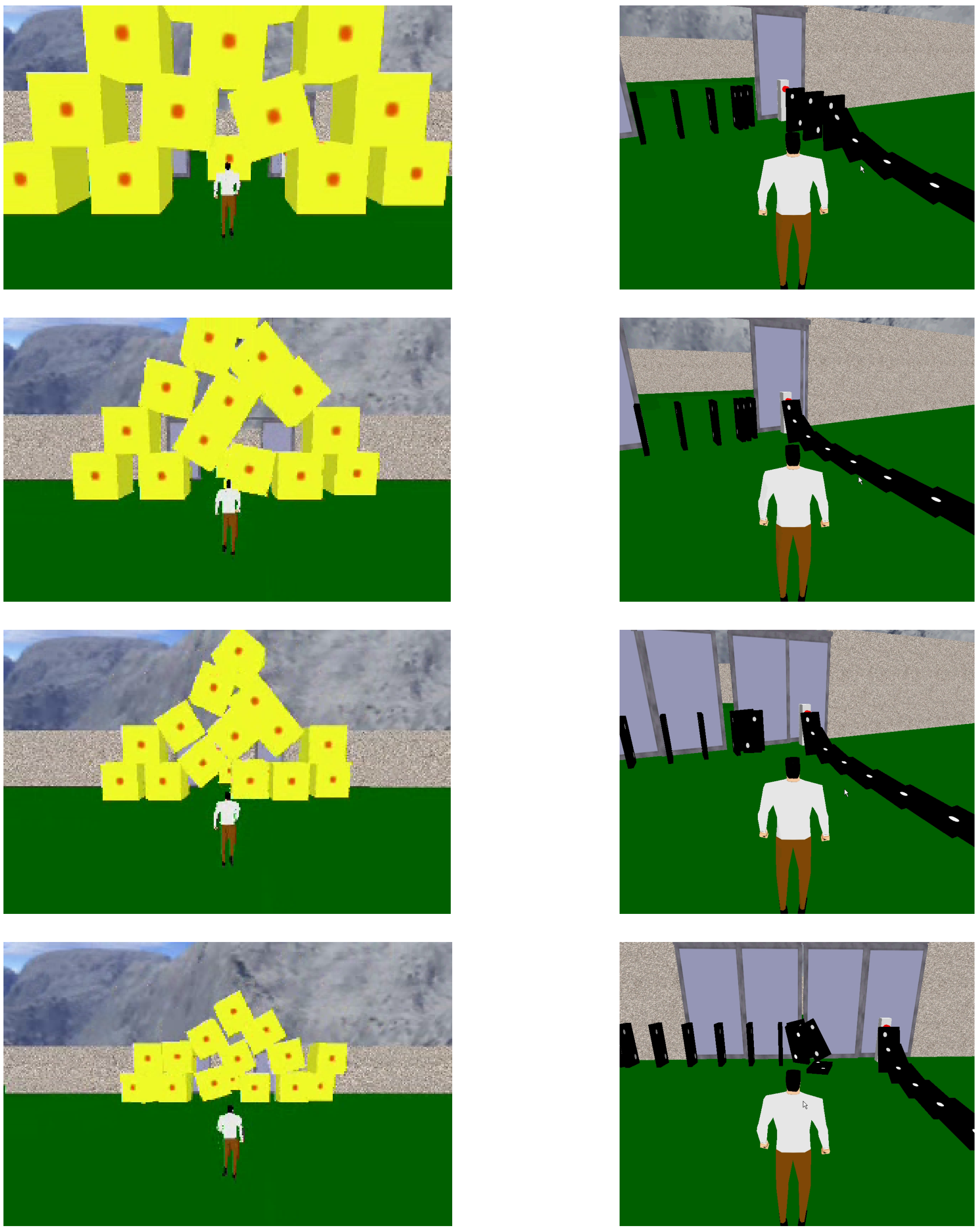


Figure 10.1: Illustrations of the new physical and interactive capabilities of the extended interactive object approach provided by the *ExtReAM* animation library.

CHAPTER 11

Conclusions

In this part, we discussed how animation and rigid body simulation can be used to improve the realism of virtual worlds. Although the interactive object platform, which was presented in the previous part, enables developers to create dynamic interactive behaviors for virtual worlds, it was clearly lacking physical realism and the ability to use dynamic animations in a general way. In order to solve this issue we delineated how we created and employed our own animation library to fill this gap.

Taking into account the evolution of graphics capable devices and the dynamic approach of our interaction system, *ExtReAM* is built around an object-oriented, platform-independent core that can easily be extended by plug-ins. While the core system provides functionality for managing plug-ins and objects, the plug-ins are responsible for more specific tasks such as object loading and various animation techniques. Different plug-ins can be used for different platforms when necessary and plug-ins need to be loaded only when their functionality is required. Furthermore, we described several plug-ins that were created in order to support the animation requirements of our interactive object framework and demonstrated them in a Desktop and PocketPC application. As a result of the design choices, *ExtReAM* offers application developers with a flexible 3D animation library that is easy to integrate, work with and extend with new animation techniques even without recompiling the core.

These properties make integration of this library into our interactive object platform almost trivial. However, in order to be backwards compatible with the basic version, application and virtual world developers can choose if they want the original simulation technique, or the new approach. In the latter case, the simulation and animation components of the interaction system employ ExtReAM instead of basic simulation and animation techniques, resulting in the above mentioned advantages. Plug-ins provide functionality for more realistic virtual worlds by supporting, amongst others, rigid body dynamics and IK. In order to describe these new features, the interactive object and world descriptions had to be extended as well. These extensions mainly include physical properties to be described as well. As we coupled the animation library's command system to interactive object system, including *ExtReAM* causes all of its commands to be available for the interactive object system as well. Consequently, this extended interaction system provides, apart from more realistic and physically enabled virtual worlds, much more interaction possibilities for VE and NVE systems

Now that we have created a realistic and dynamic interaction framework, that enables every kind of object to interact with any other object in a virtual world, and has built-in support for dynamic and on-the-fly animation techniques, it is only logical that we further investigate how this can be exploited in order to achieve a higher level of presence and participation. More specifically, although we mentioned the final link in the interaction process *avatar control*, several times, and provide built-in support for it in our interaction framework, we must analyze how dynamic animation and physical simulation can be utilized to create more realistic avatar movements and interaction techniques. Furthermore, the impact on the user's feeling of being present in the virtual world is important to be examined as well. This will be the subject of the next Part of this dissertation.

Part IV

ENABLING DYNAMIC 3D AVATAR CONTROL

Table of Contents

CHAPTER 12

Introduction

In the first part of this work, we discussed how VR tries to exploit the idea of fully immersing a user into a computer-generated environment. Kallmann argues that the concept of immersion in a VE is rather relative, depending on many factors [Kallmann 01]. Graphical realism and plausible simulation are two important factors, but the way users are represented is just as important. Seamless coupling between a user's actions and his avatar representation is a necessity if naturalness is to be achieved with respect to interaction in VEs. This is even more so in NVEs where, as we discussed in Part I, an avatar has even more tasks to fulfill. Manninen argues that the representational aspects of interaction, such as appearance and embodiment are major factors that can increase the communication, co-ordination and collaboration possibilities within NVEs [Manninen 04]. If other users can perceive a user's task and the actions he is performing, they have a chance to act accordingly.

However, although the expressiveness of avatars is relevant to interaction within the NVEs, the avatars in most contemporary NVE systems tend to be limited in terms of their expressive capabilities. Although skeleton based modeling is the most supported approach in contemporary NVE systems, the avatars are limited to displaying a restricted set of predefined keyframe animations. The main reason behind this is the fact that this is computationally inexpensive, easy to use and distribute. However, as only a limited set of animations can be provided, this approach is far from flexible, even if real-time adjustment of the animation parameters are supported (e.g. the speed at which an animation is to be displayed or blending weights). While this approach is suitable for displaying reoccurring events such as walking or running, the instant creation of situation-specific animations is impossible, resulting in the fact that avatars become customized 3D cursors that mark their owner's position within a virtual world and perform repetitive actions. Consider the case of object grabbing for example. If the animation engine relies solely on keyframing, allowing users to grab objects at different heights requires the availability of multiple predefined grabbing animations, which can cost an animator a lot of time to create and requires more memory. This problem is often circumvented by placing objects at one or possibly a few fixed heights in the virtual world and predefining very specific and detailed animations for grabbing at these heights. However, this approach clearly limits the freedom of the VE designer. Furthermore, even when these interaction restrictions are applied, the results are still not always satisfactory. An example is a user that is not perfectly aligned with the object he is picking up. As a result, in order to create truly dynamic and interactive virtual worlds, more advanced animation techniques have to be introduced in NVEs.

In the previous part, we described an animation system that permits more dynamic animations to be integrated in NVEs as well. This animation system can, apart from realistic object behavior, also be used to create more dynamic avatars for instance by employing IK. Integrating IK in NVEs can compensate for their current lack of flexibility, since it enables on-the-fly creation of new animations. As an example, reconsider the case of object grabbing. By using IK, a grabbing animation for the arm that takes into account the height at which the object is located can be created on-the-fly. It suffices to feed the position of the object to the IK algorithm, and save the current pose of the model, then a new pose is calculated by the algorithm as respectively the first and second keyframe of a new animation. This new animation can then be displayed by the animation engine as if it were a predefined animation. If appropriate constraints are applied to the different parts of the IK chain [Welman 89b], this approach generally yields more realistic results than displaying a predefined grabbing animation. Furthermore, no grabbing animations need to be predefined, and no restrictions need to be applied to the height at which objects can be located in the VE. Furthermore, if the right input devices are available, IK can also be applied to directly control the end-effector of the IK chain of an animated object (typically the hand of an avatar's right arm). However, it should be clear, that an animation system will not on its own be able to convert user's intentions into the correct representation and avatar reactions. First of all, the user's intentions will need to be captured in some way. Thus, an input device is required, that can convert user input into a usable digital format. Secondly, this input must be analyzed and converted into virtual world actions, performed by the avatar. As we discussed in Part I, this software component is referred to as an interaction technique.

Apart from converting input into virtual world actions, another important aspect of interaction at the end-user's side involves giving the user feedback on his actions. Most NVE systems therefore only rely on visual and audio rendering. However, as we discussed in Part I, haptic rendering, stimulating the sense of touch is becoming a mature research area as well. Furthermore, due to the increase in PC capabilities, it is more and more applied in practice. This resulted in several haptic IO devices, collision detection techniques and interaction methods aimed at providing users with the ability to feel the VEs that before could only be seen and heard. However, most of the work in this area has remained limited to research projects and specialized setups and have not yet reached a large audience. This can at least partially be attributed to the high requirements for haptic rendering calculations and the high costs involved with haptic devices. The prices of touch-enabled I/O devices are, however, decreasing. For example, Novint Technologies, Inc. has recently released a cheap haptic device, the Novint Falcon [Novint 07], which is aimed at the gaming market in order to replace the mouse. Already, a modification of the popular *Half-Life® 2* [Half-life 2 07] has been created which adds force feedback when a gun is fired or when the player is being shot during the game. The release of these kinds of devices could lead to the propagation of haptic technology to a larger audience. On the other hand, as haptic applications use specialized collision detection and simulation mechanisms in order to support haptic IO, they still suffer from problems with real-time restrictions such as high update rates. This results in the fact that most haptics remain limited to specialized set-ups for specific applications. A less strict approach, employing the techniques that are available in many systems, such as a rigid body simulation system, can also be used in order to generate realistic haptic feedback. While this approach might not be as exact as specific haptic routines, it will suffice for most kinds of interactions and the results could potentially form a solution for a much broader group of applications.

While the previous two Parts mostly focused on the internal mechanisms for supporting realistic and dynamic interactions in (N)VEs, this part will focus more on the interaction mechanisms provided to the user. More specifically, we will present two interaction techniques exploiting the dynamic mechanisms and animation techniques provided by our extended interactive object platform. First of all, we will discuss how dynamic IK animation can support more realistic avatar interactions in virtual worlds. As we discussed earlier, dynamic animations are often avoided in NVEs since they require more networking resources than predefined animations. Therefore, we will also elaborate on how this new interaction technique can be distributed as efficiently as possible. Secondly, we will discuss how rigid-body simulation, which is present in our interactive object system, can be exploited to generate haptic feedback forces. More specifically, as 'travel' is probably the most ubiquitous of all interactions, we delineate how we created a haptic travel method that allows users to navigate through a virtual world while receiving haptic feedback on what happens to their virtual counterpart. After presenting this haptic travel technique, we evaluate the approach in a formal experiment that analyzes the influence of haptics on the user's feeling of immersion during travel in a virtual world.

Before we get to this, the next Chapter introduces some background on avatar control in virtual environments, haptic control and its relation to physical simulation. Furthermore, we provide a more detailed explanation of presence and discuss some presence measuring tools.

CHAPTER 13

Related Work

In order to have an avatar interact with the virtual environment a user must be able to control his virtual counterpart's actions. In [Capin 98] the authors identify three broad groups of control. *Direct avatar control* enables the user to adjust the avatar's pose directly. *Guided avatars* on the other hand, can be given a specific task which the avatar will then perform, such as walk forward, but the user has no direct control over the avatar's pose(s). Thirdly, *autonomous avatars* are steered by user or AI-specified high-level goals, such as 'go to room 12' or 'pick up the glass and drink'. These autonomous avatars can extensively be used in NVEs, especially to inhabit places that might otherwise appear unattractive or empty. However, in this work, we will not focus on the use of autonomous avatars. The major advantage of direct avatar control techniques is that they allow a natural mapping between a user and his avatar's movements to be created. This increases the user's feeling of identification with the 3D virtual character. On the other hand, guided avatars are the most common, as they are easier to implement and can be used without specialized input devices for tracking the user's actions.

As a result, most interaction techniques are not concerned with avatar animation directly, but focus on two other aspects: navigation through the virtual world and the manipulations of objects [Bowman 99]. These are the standard actions that almost every (N)VE system provides. However, as discussed earlier, most of them have been developed in an ad hoc fashion as a necessity to be able to demonstrate other functionalities. Several taxonomies have also been proposed to categorize these ITs and to define their building blocks. *Navigation* is undoubtedly the most ubiquitous of the basic interaction tasks, as the user mostly wants to explore the virtual world. According to Bowman [Bowman 97] the navigation task consists of two phases: a cognitive component called *wayfinding* and a second named *viewpoint motion control*. In the wayfinding phase the user plans how he can reach the desired location, based upon an internal mental map of the environment together with several (mostly visual) cues from the environment. *Viewpoint Motion control* or *travel*, on the other hand, is the physical component used to move one's viewpoint between different locations in the VE. Both phases are separate processes, although they can have an influence on each other. In literature, several traveling metaphors can be found, either from an egocentric (first person) or an exocentric (third person) point of view [Pouprey 98, De Boeck 05]. The best known metaphor for exploring virtual worlds is the 'flying vehicle' metaphor, where the virtual camera is moved as if it is mounted on a vehicle or object which can be moved in space. The metaphor provides a very understandable and controllable solution to the user either with 3 or 6 degrees of freedom (DOF). The 'Scene In Hand' metaphor is more suitable for inspecting objects as it makes it possible to easily rotate the scene and to inspect it from different viewpoints. Finally, 'World in Miniature' provides the user with a miniature overview of the world, allowing to place the camera at an arbitrary position.

Mine [Mine 95] as well as Bowman [Bowman 98] recognize that many of the problems with understanding these 3D interactions result from the lack of haptic feedback. Therefore, it is no surprise that in the past decade there has been an increasing interest in this area [Oakley 00]. In the scope of this work we will focus on force feedback, which relates to the mechanical production of information sensed by the human kinesthetic system. However, we are aware that tactile feedback, more focused on the sense of pressure can have its merits for direct and guided avatar control as well.

To achieve natural force feedback in VEs special haptic IO devices are used. These devices usually consist of motors or brakes which have to be updated at least at 1000 Hz to avoid instabilities. Compared to the 25 to 50 Hz update rate of visual rendering and physics calculations this is a tremendous difference, which is the reason why many haptic applications are limited to the rendering of rather static environments or virtual worlds with a limited number of simulated objects.

As mentioned earlier, physical simulation has also been extensively investigated in the last few years. This resulted in several interesting applications and a number of physics SDKs, which were discussed in Chapter 8. 3D input in combination with haptic feedback has not been applied in many applications with real-time direct avatar control, certainly not in combination with physical simulation. In [Oore 02] Oore et. al. present a novel 3D input device for interactively controlling animations of a humanoid character. By embedding the trackers in a physical tubing, the authors establish a tangible interface with various coordinate frames inherent to the character, providing a valuable synergy between the physical input device and the underlying transformations from input to motion. This is only one approach, most other approaches rely heavily on motion capture employing expensive suits, or capture studios. Although these methods provide a way to control an avatar, apart from visual cues, they lack extensive feedback on what is happening to it in the virtual world. Furthermore, support for these kinds of techniques in NVEs is almost non-existent. Avatar control involving haptics is even less explored. Bierz et.al [Bierz 05], describe haptic interaction with cognitive avatars in their CONTACT project. More specifically, the user can interactively influence an actor's planned path by blocking the avatar's path with his virtual hand, forcing the actor to calculate a new path. While interacting, the user receives force feedback from the collisions. Several other authors, such as [Magnusson 02, Nemec 02] describe navigation and object recognition schemes in VEs for visually impaired people. The virtual worlds that are used, are fairly static and there are no dynamics or animations involved. Also, due to severe haptic rendering and output requirements, the number of objects is usually limited in these environments. De Boeck et al. [De Boeck 02] describe navigation using the 'Camera In Hand' metaphor, similar to the 'flying vehicle' metaphor, enhanced with a bounding box which the user could feel while navigating. They compared their navigation technique by using a PHANToM and a spacemouse as alternative input devices. The usability study showed that users equally liked both input devices and the tasks were performed equally well. This study, however, lacks an investigation on the user's presence and workload.

The concept of *presence* has been defined by several researchers from different areas in different ways. In this work, however, we are only concerned with the definition in the context of interactions in 3D VEs. Presence can then be defined as: a state of consciousness, the psychological state of being there [Slater 97, Hendrix 96]. Similarly, Witmer and Singer [Witmer 98] define presence as the subjective experience of being in one place or environment, even when one is physically situated in another. Two psychological concepts are of interest concerning presence, those are *involvement* and *immersion* [Witmer 98]. As users focus more attention on the virtual reality stimuli, their involvement in the virtual reality experience increases, resulting in an increasing sense of presence. Immersion, on the other hand, depends on the extent to which the stream of stimuli and experiences that the VE provides make people feel included in and able to interact with the environment. Factors which affect immersion include isolation from the real environment, perception of self-inclusion in the VE, natural interaction and control, and the sense of self-movement [Witmer 98].

Investigating a participant's perceived presence is far from straightforward and a lot of research has been done in trying to determine a useful methodology for measuring it. Three categories of measurement methodologies have resulted from this: subjective, behavioral and physiological [Insko 03]. *Subjective methods* rely on self-assessment by the participants, and include answering questions such as "How real did the environment seem to you?". These self-assessments are normally performed after a task has been performed in the virtual environment. Several questionnaires exist: the most employed being Witmer-Singer [Witmer 98] and SUS [Usoh 00]. The advantages of these questionnaires are that they are specifically tailored for what they want to measure, and they are easy to use, validate and interpret. They also do not break the presence as they are conducted post-immersion. This is immediately also the main disadvantage: as they are post-immersion the user answers them after the experience of presence while the experience could vary according to time or might be better or worse near the end of the experience. *Behavioral methods*, on the other hand, take a totally different approach, and examine actions or manners exhibited by the user that are responses to objects or events in the virtual environment. For example, does the user duck if a virtual object is thrown at his head. The main problem with behavioral methods is experimenter bias. As he is aware of the experimental conditions, it is hard to know if the observed behavior is a consequence of an experimental condition. Finally, *physiological methods* attempt to measure presence by gauging changes in the subject's heart rate, skin temperature, skin conductance, breathing rate, etc. These methods are, compared to the former, objective and continuous but harder to use. It takes time to place the sensors and it is harder to interpret the results. Meehan [Meehan 01] created a 'pit' room containing a virtual 20-foot precipice to induce stress in the users and measured heart rate, skin temperature and galvanic skin response.

Presence with regard to VEs has mostly been investigated for immersion in setups in which only the visual and sometimes the auditory sense is presented to the participant [Slater 97, Witmer 98, Meehan 01]. Sallnäs et al. [Sallnäs 00] have investigated the presence in collaborative haptic environments. They concluded that haptics improves task performance, perceived task performance, and perceived virtual presence in the collaborative distributed environment. Adding haptics to VEs does, however, not only influences presence, it can also influence the workload which a user experiences. Oakley et al. [Oakley 00] investigated several haptic effects of which some had a higher workload than others. They used the NASA Task Load Index (TLX) questionnaire [Hart 90]. NASA TLX is a subjective workload assessment tool, which allows users to perform subjective workload assessments on operator(s) working with various human-machine systems. NASA TLX is a multi-dimensional rating procedure that derives an overall workload score based on a weighted average of ratings on six subscales:

- mental demand: how much mental and perceptual activity was required;
- physical demand: how much physical activity was required;
- temporal demand: how much time pressure did you feel due to the rate or pace at which the tasks or task elements occurred;
- effort: how hard did you have to work (mentally and physically) to accomplish your level of performance;
- performance: how successful do you think you were in accomplishing the goals of the task set by the experimenter;
- frustration: how insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during the task.

The following chapter discusses an interaction technique that directly controls the avatar's pose using IK and a 3D input Device. Thereafter we propose a method for employing a rigid body simulation system to calculate haptic force feedback. This is then demonstrated by creating a haptic travel technique that gives force feedback on what is happening to his virtual representation. Finally, we investigate what impact the haptic information has on user's task performance, feeling of presence and workload in Chapter 16.

CHAPTER 14

An Avatar Control Method for Direct Interaction

14.1 A Dynamic Physical Avatar Object

As we extensively discussed in the previous Chapters, our interactive object system generalizes all objects in the virtual world. Hence, if we want our animated avatar to be able to interact with other objects, we need to create an interactive object that incorporates this avatar. Once this object is created, a controller can be implemented that communicates with this object in order to move its parts and animate the model to display its actions and have resulting interactions in its surroundings. Since we will rely mostly on the physical interaction aspects, our avatar object will have to contain a physical representation and physical actions as well. We must emphasize that although this avatar and its interactive object are only an example, the underlying techniques are kept as general as possible. It is however required for each other (kind of) animated character to create an associated interactive object, that fits the graphical representation. In case the character is very different and implements completely different commands for moving according to its graphical representation, it might also be necessary to implement a different controller. However, for most avatar types used in current NVE systems and MMORPGs this will not be necessary.

The basic properties of the avatar's interactive object consist of two parts. The first part incorporates the 3D animated character like the one shown in Figure 14.1(a). Although this model is quite simplistic in terms of mesh quality, it suffices for our purposes. It has integrated keyframe animations for walking, running, jumping and standing still. The animations are defined on a skeleton consisting of 21 bones, also shown in the Figure. Since we do not require complete physical interaction with every part of the body, we provide the object with a less detailed physical representation consisting of a flattened cylinder surrounding the human model. This more efficient physical object suffices to make sure the visual representation does not run through walls etc. As a second part, we define a spherical object, that will be coupled to the hand of the avatar when his arm will be transformed with IK with physical joints. To avoid any more complexity, we did not add other physical objects to follow the entire arm. While this would result in more realistic behavior when the arm would hit something, we are in this interaction technique only concerned with the ability to use the hand as a means of interaction with the environment, for which this physical representation suffices. The sphere's connective joints, specified in the interactive object description, restrict it from moving outside the animated model's reach by using the same limitations as specified for the model's skeleton joints.

Apart from the parts and physical objects, the basic object properties indicate the commands that allow the object to move forward, backward, left and right and state variables that contain if IK is enabled. Furthermore, we define the IK end goal position as a state variable and another variable that points to a separate file describing the IK chain that will be used in the interaction technique. First of all, this file indicates which joints of the model form the IK chain. Secondly, it describes a bounding volume which the end of the IK chain can never leave when it is being animated through inverse kinematics. And finally, DOF constraints can be specified for all joints in the IK chain. These constraints will be taken into account by the IK algorithm when it is calculating a new pose. This way, we can prevent joints from rotating into positions that are physically impossible to achieve. The separation of IK information from the interactive object data model is done for reusability reasons. This is specifically useful for similar kinds of avatars, which often use the same underlying skeleton, and only have a different skin mesh. As an other advantage, we can use the same animated model to create a

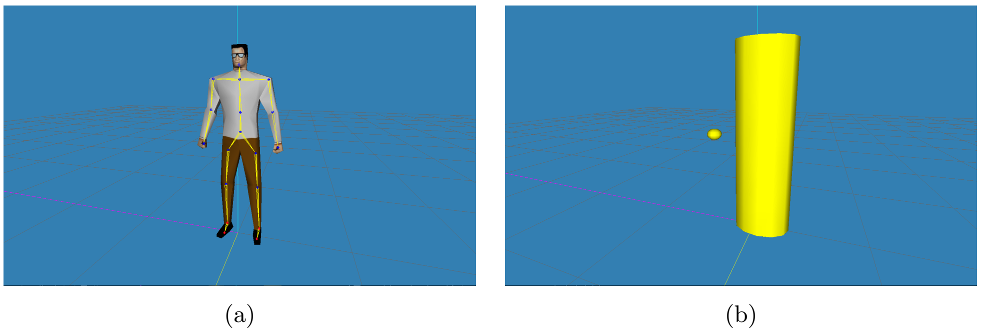


Figure 14.1: (a) The avatar's 3D animated model and (b) the physical representations of the parts.

left and right handed interactive avatar, by changing the IK chain. In the case of this avatar, the IK chain consists of the joints ranging from the shoulder to the hand.

As *ExtReAM* is used, employing the IK, keyframe and physics plug-ins, all the commands that are supported by them will be available. The interactivity properties of the object description furthermore describe several important functions: `walkForwardToPosition`, `walkBackwardToPosition`, `RotateDegrees` and `moveIKChainToPosition`. These functions are described in scripts and combine animation engine calls consisting of the application of physical forces (to move the entire physical avatar), partial running of keyframe animations (for example to let the graphical representations walk while the physical representation moves) and to apply IK on the specified IK chain in order to move the hand to the goal position. We hereby note that we do not focus on human models only, but in contrast provide a solution that is applicable to all kinds of animated avatars.

14.2 A Direct Avatar Controller Exploiting IK

The next step in creating a direct avatar control technique that interacts with the environments is the implementation of an interactive object controller, that communicates the user's intentions to the object. We emphasize that regarding the interface, this work focuses on desktop VR systems. So in a first approach, we created a controller that employs the keyboard as a way to control the avatar's physical end of chain representation and its IK chain.

More specifically, the controller takes as input key presses, which trigger the `moveIKChainToPosition` behavior within the IK area. This results in a physical movement of the end chain object in the direction of the new end position, using the rigid body simulation plug-in to make sure it does not move through other objects. The new position of the physical end chain object is then used as the goal position for the IK algorithm. The IK plug-in of *ExtReAM* then instantly calculates a plausible pose for this chain taking into account the skeleton's DOF and damping factors. In this way, the controller can be used generally for all kinds of controlled animated interactive objects. In the case of our interactive avatar, this results in the avatar's hand moving according to the input of the user, unless the physical object of the hand is colliding with another object. Moving the physical representation of the hand is achieved through applying a force to that object, which means it can be used to push other movable objects away, or it can be pushed away by other objects as well, depending on the masses and forces that are applied on those objects.

Although physically and graphically the results of the proposed technique proved to work correctly, the use of the keyboard as the input device for controlling the arm proved inefficient and counterintuitive. First of all, as the keyboard is a 2D input device, it is inefficient to be used to directly control an object in 3D space. Secondly, apart from the visual feedback that is given to the user (the graphical movement of the avatar's arm), the user receives no reference information of where his virtual hand is positioned, resulting in a low level of identification with the avatar. Finally, this control also requires a context switch, as the keyboard was also used to control the avatar's navigation through the scene. Thus, users explicitly had to switch between direct hand/arm control mode and travel mode, which in informal tests often resulted in situations where users tried to move forward while they were actually controlling the arm and vice versa.

As a solution to these problems, we implemented a second dynamic avatar controller. Its functionalities are exactly the same; however, instead of taking input from the keyboard, it allows input from a 3D input device. The MicroScribe-3D input device was chosen for tracking the user's gestures. Although it was more specifically designed for digitizing real world models into 3D virtual models, it provides an affordable and efficient solution for our purpose as well. Furthermore, it is very lightweight and supports a large workspace. Based on the observation that the MicroScribe-3D is constrained in its movements just as the IK chain is, we automatically map the allowed workspace of the MicroScribe-3D onto the IK bounding volume of the model. This mapping is model-dependent, as each interactive object with IK chain

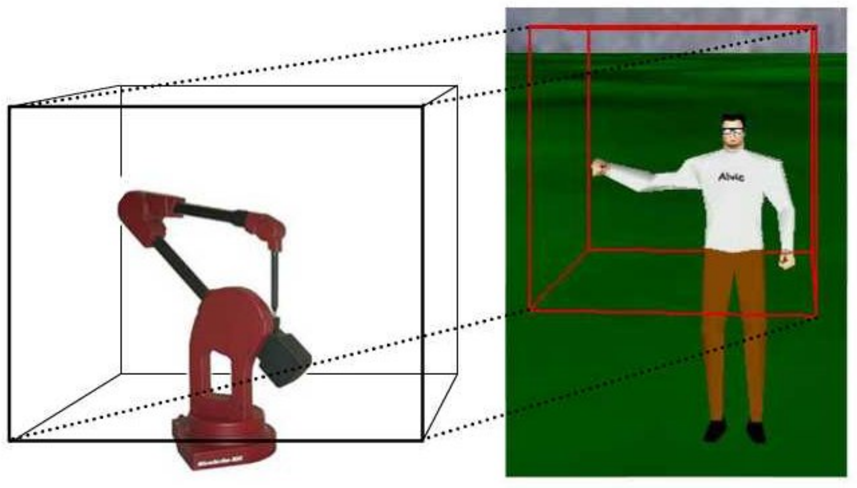


Figure 14.2: A schematic representation of the mapping of the MicroScribe-3D input field onto human avatar's the IK chain workspace.

specifies its own IK workspace. This ensures that the input position from the MicroScribe-3D is always correctly transformed to the corresponding relative position in the IK bounding volume of the interactive object. This position is subsequently fed to the physical representation, whereto the IK algorithm will try to move the end-effector as close to as possible. A schematic representation of how 3D input with the MicroScribe-3D is mapped onto the IK described workspace is given in Figure 14.2.

14.3 Distributing Animation Information in NVE Systems

Animating articulated objects in NVEs introduces new network traffic since animation related information will need to be distributed among connected users. There are several ways animation data can be represented for transmission over a computer network [Capin 98]. Some of these representations place a large load on the network, but do not require a lot of processor time to encode/decode. Others are computationally expensive to encode/decode, but require only small update messages. Since both network bandwidth and processor time are scarce resources, NVE designers will need to make a trade-off and decide which of these two resources is more valuable to their application and architecture.

As the ALVIC framework, which we use for testing, already consumes a lot of bandwidth by employing video avatars, we decided to keep the new network traffic introduced by animation as low as possible by using compact update messages. This decision is also based on the observation that there have been steady increases in the clock speed of processors, while Internet connections have not improved accordingly the past few years. We believe this trend will persist at least in the near future. Therefore, when keyframing is used to animate a model, only a high-level description of the model's active animations is sent over the network. When receiving such a message, remote sites must start interpolating locally to be able to display these animations as well. Similarly, instead of transmitting the transformations of all the joints in the IK chain when animating using inverse kinematics, only the position of the goal position of the chain is sent. Upon arrival of such a message, remote hosts will need to calculate the transformations of the intermediate joints in the chain themselves by feeding the received goal position to their animation component. It should be apparent that this approach minimizes the network load but increases the necessary processor time in the decoding phase.

A simple calculation shows the bandwidth reductions our approach is capable of achieving. Suppose the IK chain of a model consists of N joints. To precisely represent the orientation of a joint, one float triplet is required (joint lengths are fixed, thus positions are calculated from the orientation of the joints). Furthermore, suppose animation updates are transmitted at a rate of 25 per second, an update rate that should be achieved by our platform on PCs. If we included the transformation of all IK joints in each animation update packet, a bandwidth of $N * 3 * floatsize * 25$ would be required per second. Our approach reduces the bandwidth usage by a factor of N since only the position and orientation of the end-effector is transmitted in every animation interval. Note also that the benefits of our approach increase proportionally to the number of animated models present in the virtual environment. Consider the case of an interactive arm containing 4 joints in the IK chain. Our approach would require 1 * 3 * 4 bytes * 25 = 300 bytes/second. The normal approach would require 1200 bytes/second (ignoring headers of course), so we decreased our bandwidth usage with 900 bytes/second per interactive actor.

For keyframe animations, we take an even simpler approach. We just send start and stops for animations such as walking, resulting in the fact that only one small message needs to be send instead of a triplet of floats per joint per frame, resulting in an even greater bandwidth reduction. However, there is also a downside to these traffic reductions. While our approaches minimize the network load, a reasonable amount of processor time is required to decode incoming animation messages. Furthermore, there is also a synchronization loss associated with distributing animation data this way. However, this is acceptable for most multi-user applications as the most important aspect of the animations is to show the user and other participants of the NVE where the avatar is, and what actions it is performing. As an example of this synchronization error, consider when keyframe animation is used for showing that an avatar is traveling through the virtual world. With this approach, the exact frame of the walk animation will not be identical on all other clients, but it will show the walk animation. Showing the fact that the avatar is walking suffices, and exact keyframe synchronization of the animations is not a necessity, especially weighting it off to the extra network usage it would require. Similarly, for the IK chain, since only the end-effector position is distributed, the end position of the IK chain will be the same on all sites, but the remaining joints are calculated locally. Thus these can be different. However as we are mostly interested in the hand position, we are not really interested in the other joints. Furthermore, as the same IK algorithm is used on the different sites, and the goal position is updated every frame, the results will mostly be very similar. However we realize, that for some specific applications exact avatar pose synchronization might be a requirement. In that case, complete skeleton poses must of course be distributed every time frame. We are aware that several solutions exist to even further minimize the number of update messages, such as dead reckoning techniques, however these network improvements are not the subject of this work, so we will not discuss them here.

14.4 Results

The dynamic avatar controller and MicroScribe-3D input device were added to the ALVIC test application that we also used for earlier tests. The mapping of the input device's workspace on the humanoid avatar's IK workspace in combination with the device's positioning together with the skeleton limitations, resulted in the fact that the pose of the avatar's arm approximately corresponds to the pose of the user's real arm when he is using the device. Figure 14.3 shows some of the results of this mapping with a humanoid animated model. As a result, this approach becomes a very useful and intuitive control method over the IK chain, which confirms the view given in [Hinckley 94] and [Mine 97] stating that the inclusion of motion relative to a spatial reference or the user's own body in interaction techniques decreases the cognitive load of the user. Furthermore, the switch between controlling the arm and navigation also became much more natural by introducing the MicroScribe-3D, since both controls are now performed by separate input devices. This also creates the possibility to control both at the same time. *ExtReAM*'s `SkeletonBlender`, then overwrites the IK chain's keyframe information by the information provided by the IK controller. Informal usability tests showed that the loose synchronization over the network worked correctly and fast enough. The possible differences in the arm's pose on different sites were not noted by the participants. This is probably the result of the fact that the focus was on the hand position, and the interaction of the hand with other objects. Figure 14.4 shows some screenshots from inside the NVE. Two avatars are working together in the virtual world. One user closes the door by controlling his avatar with the dynamic hand interaction technique. At the same time the other user pushes a table through the doors with his avatar.

It should be noted that the results presented in this section could have also been achieved by using other animation techniques, such as inverse dynamics. Some of these techniques are even capable of producing visually more realistic interaction animations than IK. Unfortunately, these techniques also have a much larger computational cost associated with them. Since processor time is still a scarce resource, at the moment IK is the most suitable animation technique to enable complex, run-time interactions for all animated models in a virtual world.

A disadvantage of the proposed interaction technique and input device is that it does not generate force-feedback information, making it possible for the user to keep his hand moving while the avatar's hand is stopped by a collision. This might result in problems with understanding the interaction method, as was also previously found by Mine [Mine 95] and Bowman [Bowman 98]. In the remainder of this part, we will discuss a method that will also provide haptic force feedback. Instead of utilizing specific haptic rendering and collision techniques, which are specifically aimed at working at haptic rates, we will discuss how standard rigid body simulation as we have in *ExtReAM's* `Physics` plug-in can be used to calculate haptic force feedback. As a result, we use information that is already available but can result in more complex and dynamic haptic scenes.

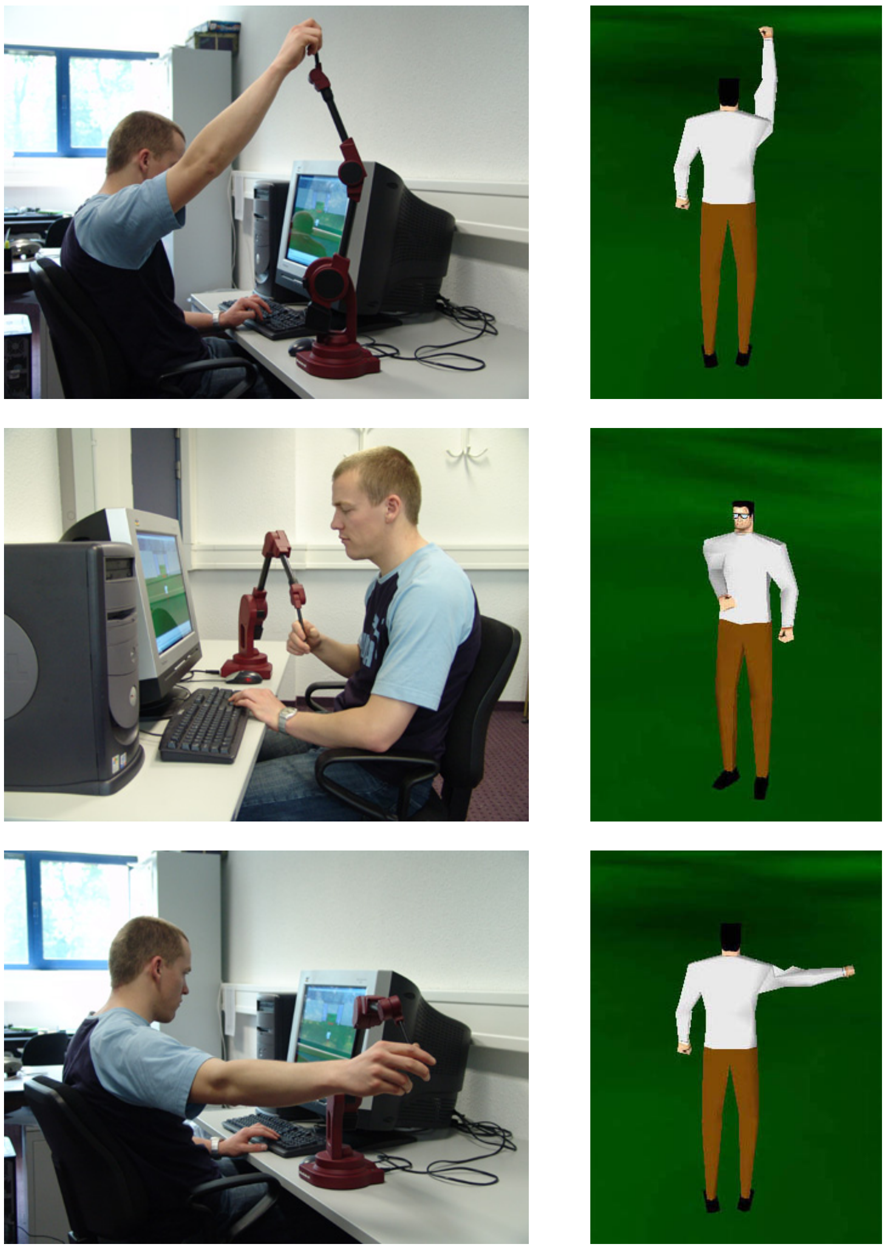
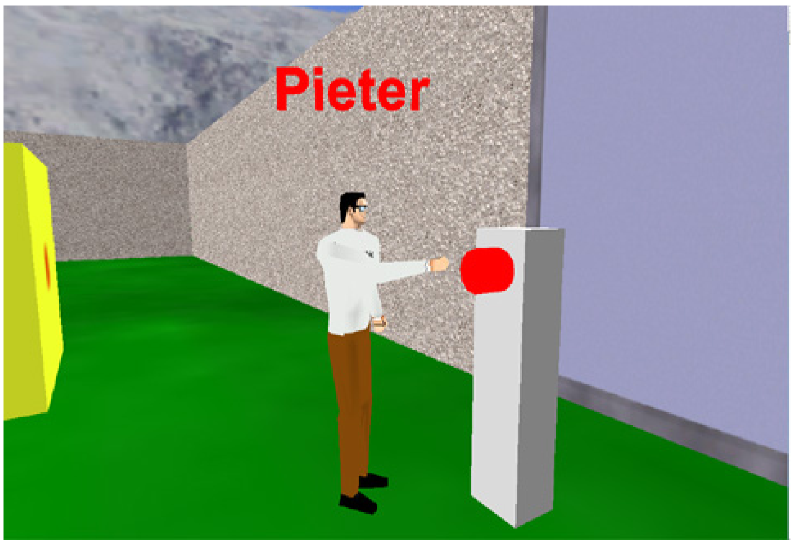
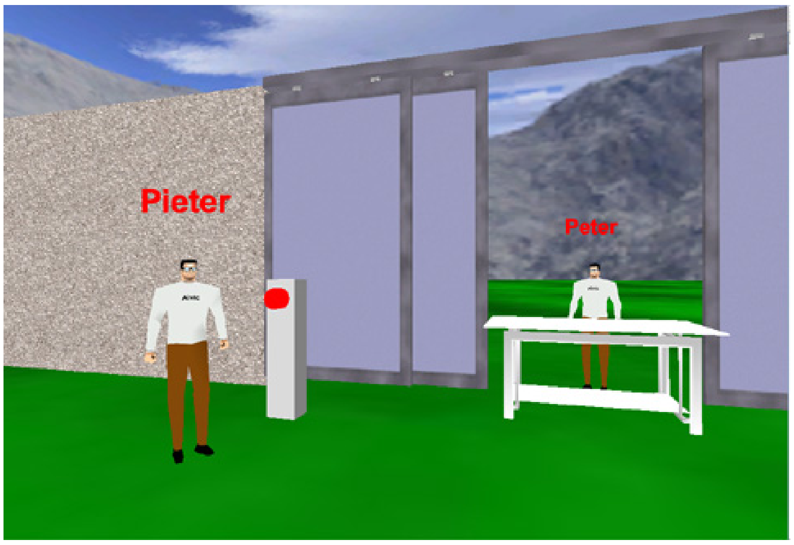


Figure 14.3: Results of the dynamic hand interaction technique. The avatar's arm is moved according to the user's input taken from the MicroScribe-3D.



(a)



(b)

Figure 14.4: Screenshots from inside the physical ALVIC system as seen from one of the users. (a) A user utilizes the MicroScribe-3D arm control to push the button triggering the door to close. (b) As another user is pushing a table through the door, the closing of the doors is stopped and the box is now stuck in between.

CHAPTER 15

A Physical Travel Method Enabling Haptic Force Feedback

15.1 Creating A Haptic Controller by Employing Rigid Body Dynamics

15.1.1 Requirements

To realize a stable coupling between user input, physical simulation and providing force feedback to the user, several components must are required. First of all, an object must be present in the virtual world that can be controlled by the user. Since travel is the most employed interaction technique, we aim at providing a realistic method for it. We can reuse the interactive avatar object we used in the previous chapter, ignoring the hand part as it is not relevant for travel. Secondly, a simulation component is required, to make sure collision and force information is given to this object to let it move in a believable and realistic way. As we explained in the previous Part, the `Physics` plug-in of the *ExtReAM* framework was specifically designed for this purpose and can thus be utilized for this task. Finally, we we need a haptic output device and a conversion routine that translates the collision information from the simulation into real-world forces that can then be displayed to the user.

There is however one issue with our components. As already mentioned, in order to give the user stable haptic feedback, haptic rendering requires a high update rate, in the order of 1000 Hz. This is why most haptic applications utilize specialized collision detection routines, which can operate under these heavy real-time requirements. However, if we look at our simulation component, we see that the update rate is significantly slower, 30 to 50 Hz, depending on the requirements of the application and the numbers of objects that need to be simulated. How we circumvented this issue is delineated in the following sections, where we also discuss how a haptic controller plug-in was created.

15.1.2 A Haptic Component for ExtReAM

Although the haptic component is not necessarily bound to animation, our solution relies heavily on rigid body simulation, therefore, we can implement it as a separate plug-in of the *ExtReAM* framework, obviously dependent on the `Physics` plug-in we outlined in the previous Part. In order to allow it to send force feedback information to the users, we integrated the low-level HDAPI from the Sensable OpenHaptics^{TM} toolkit [SensAble 07] as an interface to the haptic device. For grabbing input and displaying haptic output, we employ a PHANToM device [Massie 94].

How the `HapticController` plug-in was constructed, and how its components relate to other plug-in components and the application is illustrated in Figure 15.1. The main component of the plug-in, the `HapticController`'s *actor* is used to initialize the plug-in and forms the link between the haptic rendering and the physical simulation. Its main tasks are converting the input into physical movements for the physical object that is controlled and providing the user with force feedback based on the physical simulation of that object, in the virtual world. The actor has just like any other plug-in, a *step* function that is called every animation timestep by the `AnimationSystem` and is dependent on the `Physics` plug-in, that handles the physical simulation and contains the physical data and objects in the scene. Furthermore, the plug-in makes use of the `HapticDeviceManager` handling the haptic device input. It allows for information such as current position, orientation and forces from the device to be requested. In addition, it is the interface through which we can output feedback forces. The `HapticDeviceManager` is also the component that solves the real-time requirements of the haptic de-

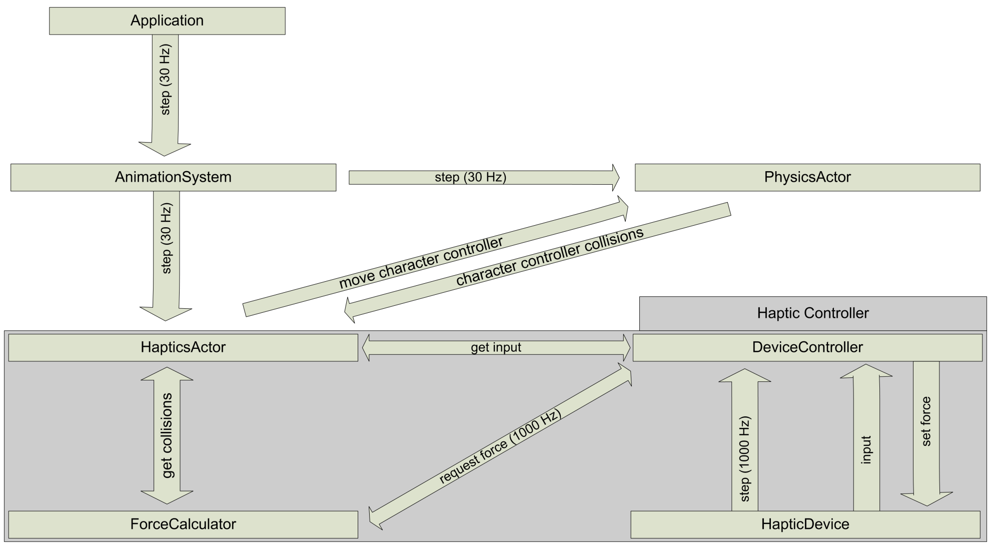


Figure 15.1: An overview of the haptic controller plug-in components and their relations to other components and the application.

vice. It runs a separate thread that is updated at the device's required update rate (at least 1000 Hz for the PHANToM). Furthermore, it is responsible for handling haptic device callbacks as well (too strong forces, overheated motors,...). To calculate the feedback forces based on the physical simulation, we implemented a base class for a `HapticForceCalculator`. When an application wishes to provide haptic feedback, it implements a derived class from the `HapticForceCalculator` class, implementing the `requestForce` function, and the `HapticDeviceManager` will apply the updated forces, from this module, at every haptic iteration. This design choice is in accordance with the design approach of the *ExtReAM* framework, and makes our plug-in as flexible as possible. As a result, one can implement several different force calculator classes and use the one that suits the application, or write one force calculator that combines several others. Also, the `HapticDeviceManager` could be easily replaced to work with other devices without the need to change the force calculator. Finally, the `HapticsActor` contains a reference to the physically controlled object. This is a rigid body, which in this situation will be the part describing the avatar's collision object as described in the previous Section. As the graphical representation will follow the displacements of this object, we refer to it as the character controller.

15.2 Enabling Physical Travel

15.2.1 Motion Control

As it is the most applied metaphor for travel, we chose to adopt an adjusted 'Flying vehicle' [Ware 90] metaphor, also known as 'driving vehicle', where we apply the rigid body simulator's gravity to keep the avatar's physical representation on the ground and attach the camera to the avatar, following its exact movements. We can use it either in a first person (egocentric) or a third person (exocentric) view. This metaphor is one of the most natural for guiding an avatar through a 3D environment and it allows the user to see exactly how his representation is moving through the VE.

There are two key parameters that define a user's movement at any time [Mine 95]: *speed* and *direction of motion.* Both have to be controllable by the input device. Since haptic devices usually have 3 or 6 (e.g the PHANToM) DOF, we have several possibilities. Logically, when the device is in the neutral position, specified by the user's rest state, no movement is applied. The most natural movement input in this case is to move the avatar in a *hand directed* way [Mine 95], meaning that the avatar is moved according to the hand movement of the user. Directly coupling the avatar's position to the end-effector position of the input device, as was done in the IK technique, will not provide a decent solution. It would imply that a direct mapping can be created between the virtual world and the device's workspace dimensions. This solution would suffer from several issues. First of all, even if the world would be relatively small, so are most of the input device's workspaces. The user would need to have a very stable hand, as each involuntary movement would result in movements of the avatar. Furthermore, in large VEs, the travelling speed caused by a small hand move, would be uncomfortably large. Also, the user would be responsible for keeping the avatar on the ground plane, unless the Y-axis input would be ignored. As a result, adopting this strategy would make it very hard for the user to navigate and explore the virtual world, and would certainly increase user disorientation in certain situations.

Therefore, we have chosen to control the avatar (and hence the virtual camera) in a relative manner, by pushing against the boundaries of a 'virtual box', more or less as described by Anderson [Anderson 98]. In contrast, we do not use a strict virtual box, but a spring force field instead. The spring force is computed by applying Hooke's Law:

$$\vec{F} = -k \cdot \vec{x} \tag{15.1}$$

Where k is a stiffness constant and x is a displacement vector. The spring is attached between a fixed anchor position (the neutral position p_0) and the device's current position (p_1). The displacement vector is such that the spring force is always directed toward the neutral position.:

$$\overrightarrow{x} = p_1 - p_0 \tag{15.2}$$

The force felt is called the restoring force, since the spring is trying to restore itself to its rest length. The stiffness constant k dictates how aggressively the spring tries to restore itself: a low stiffness constant feels loose, whereas a high stiffness constant feels rigid. To provide more flexibility, we allowed for different values of k to be used for the 3 dimensions. so for example, we can make the spring in the X-axis stiffer than for the Z-axis making sideway moves harder than walking forward or backwards.

To determine the avatar's speed, we use the length of the displacement vector $\overrightarrow{x}$ in its sideway (X) and forward component (Z). The further away the input device's end-effector is from the neutral position, the higher the avatar's velocity. As a result from the spring force, faster movement will also require stronger force input from the user. This is similar to the real world, where faster walking or running also requires more effort. Since the user's hand is never completely still, we introduce a small threshold that needs to be exceeded before the avatar starts to travel. For reasons of clarity we excluded this in the force functions here, but it can be seen as the rest length of the spring. The Y-axis force inputs could also be used, for example to let the avatar jump (if Y position or force is above the jump threshold) or crawl (Y position or force below the crawling threshold), but since we are for now only concerned with travel it is not applied at the current stage.

In order to specify the avatar's orientation, we use an approach similar to the one proposed in [De Boeck 02]. We use the direction of the input device's end-effector orientation for determining the orientation of the avatar. As long as the input is not rotated, the avatar stays facing in front and moves in the direction it is pushed. If the input device is rotated, in the test setup this means rotating the PHANToM's stylus around the Y-axis, the avatar starts rotating as well. The larger the difference between the neutral angle and the current angle, the faster the avatar rotates. As a result, the user can adjust the turning velocity to the situation. To avoid involuntary rotation, we have also introduced a threshold angle that has to be exceeded before rotation of the avatar begins. In the current stage, we only utilize the Y-axis orientation, which is the only one required for travel on a surface where gravity is applied. The Z and even X-axis angles could also be used for altering the camera's tilt

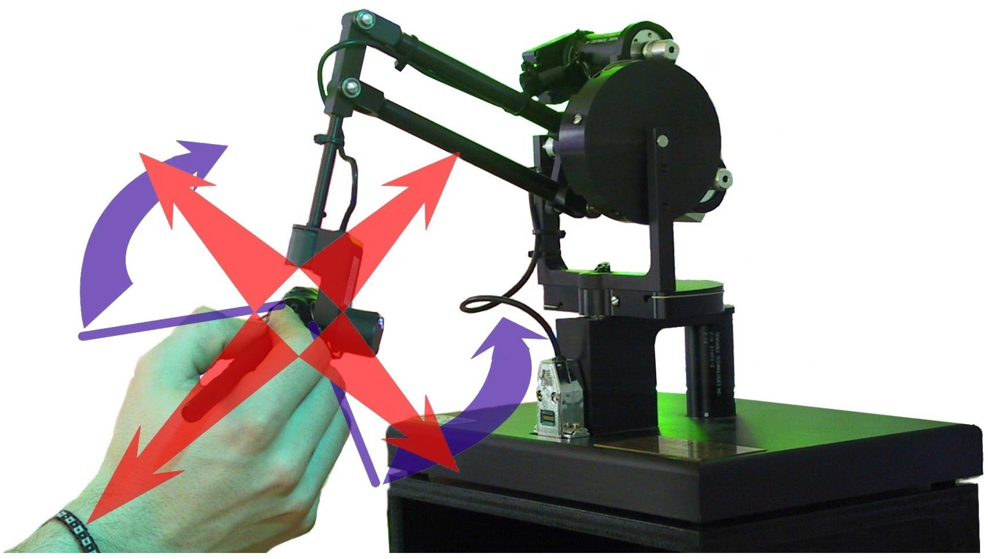


Figure 15.2: The PHANToM haptic IO device and the haptic travel control possibilities

angle, but since we are focusing only on travel, it is not considered. Figure 15.2 shows the PHANToM and its travel possibilities in more detail. The calibration of the neutral position and orientation can be done by holding PHANTOM's stylus button for more than three seconds. We do not provide the user with any other than visual feedback when rotating the avatar, since it already requires some effort to twist the arm, and we do not believe much is gained by applying extra force feedback. Also, very few desktop haptic IO devices support torques.

15.2.2 Generating Travel Force Feedback

Several kinds of forces tend to influence someone that is traveling in the real world: gravity, wind, collision responses from objects that he hits or that hit him, ... In the haptic travel technique, we try to simulate some of the most important of these forces and convert them into force feedback for the user. The calculations are performed by the `HapticTravelForceCalculator`, a derived class from the `HapticForceCalculator` class we described earlier in this Chapter.

The first kind of feedback concerns the terrain *surface* on which the humanoid avatar is traveling. People tend to walk slower, or have more trouble to travel at the same speed, on terrains like mud or sand. We simulate this by by applying a factor k_{su} to the force field, where $k_{su} > 1$ for difficult terrains. This factor is computed by requesting surface material information from the physical scene. As an extension to this force feedback, we could also provide the user with some tactile feedback in order to make him feel the structure of the surface. But since this is not the subject of this study, we will, for now, let the user trust on the visual and force feedback rendering.

A second form of travel feedback provided to the user, is based on the *slope* of the terrain he is walking on. It determines a force upward or downward, together with another factor k_{sl} for the stiffness in the direction of the avatar's movement. On a horizontal plane this latter factor is 1 and it increases/decreases when walking uphill/downhill. The slope of the ground surface is computed from the collision information that is given to the rigid body representation by the `Physics` plug-in. Also, the physical character controller has a maximum slope it can walk on. We therefore map the current slope factor between 0 and 2, so the counterforce doubles when the avatar travels up the maximum slope of the physical character controller. Any angle above this maximum results in the maximum feedback force, so the avatar cannot walk up that slope and the user can no longer push into the direction of that slope. As there is no maximum downward angle, the factor can become 0. As another result, one can walk downhill with much less force at the same speed, or at a much higher speed with the same input force then he can walk uphill. In total, the movement force feedback (in X and Z-axis) calculation becomes:

$$\vec{F} = -k_{su} \cdot k_{sl} \cdot k \cdot (p_1 - p_0) \tag{15.3}$$

A second slope-dependent force pushes the input device and therefore the user's hand into the direction the avatar is traveling (in the Y-direction) and is dependent on the length of the current movement. It is thus an addition to the normal force we use to keep the haptic device's end-effector in the neutral height. It takes into account the avatar's current speed and a range adjusted slope factor:

$$\vec{F_{slY}} = (k_{sl} - 1) \cdot |movement_{XZ}| \tag{15.4}$$

where $k_{sl} - 1 > 0$ when the avatar travels uphill and < 0 when walking downhill. The total force function for the Y direction then becomes:

$$\vec{F_Y} = -k_Y \cdot (p_1 - p_0) + (k_{sl} - 1) \cdot |movement_{XZ}| \tag{15.5}$$

The resulting force directs the user's hand up when traveling uphill and down when going down a slope in combination with the restoring force. The upward/downward force increases linearly with the increase in speed. As a result, when moving his hand away from the neutral position, on a constant slope, the user's hand is smoothly pushed up/down depending on the slope in the direction of the movement, giving the effect as if the neutral Y position smoothly moves up/down.

Finally, we give the user feedback when he walks against a wall, or when he gets hit by another object. The collision information is given by the rigid body simulator through callbacks, giving the point of collision and the collision normal. When a user walks against a wall, we just slowly (in the sense of the haptic rendering) increase the force in the opposite direction of the collision to a maximum. As a result the user's hand gets pushed to the neutral position, ending the travel requests to the physical character controller in that direction. When trying to move in that direction again, the spring force stays at a maximum value, making sure that it is no longer possible to move into that direction and giving the feeling of touching a wall. In the other case, when another, dynamic object hits the avatar, the collision information is converted from a physics hit into an impulse force in the same direction as the colliding object. When a heavy object hits the physical avatar, this will usually result in a movement of the user's hand, changing the movement of the avatar, similarly to a realistic situation.

15.2.3 Force Feedback Issues

Several issues needed to be resolved in order to combine rigid body simulation with haptic rendering. A first and most important problem involves the difference in speed between the haptic and physics simulation loops. The physics loop runs at a framerate of approximately 30 to 50 Hz while the haptic thread requires 1000 Hz updates, depending on the device at hand. This difference can lead to synchronization problems which can lead to instabilities. We avoid these by not using a direct coupling between the input device and a rigid object, which is the traditional approach. Instead, we influence the feedback force field according to the changing circumstances in the physical simulation. The `HapticsActor` operating at the physics update rate sends the relevant information about the input to the rigid body simulator at the rate it can handle. The `HapticForceCalculator` on the other hand, calculates the feedback forces at the haptic loop rate, using input data that is available at the haptic rate. Since the normal spring feedback is only dependent on the relative position of the input device at each haptic frame, and it is thus not dependent on the simulation of the physically simulated world, this feedback keeps perfectly stable in all cases.

To get stable collision and terrain feedback forces, we had to take a different approach since the collision information is only updated at the simulation rate. We resolved this by keeping the collisions valid for the duration of the entire physics frame. This does not provide a problem for force feedback, since the visual framerate will never exceed the physical rate. Also the reaction speed of the user's hand will not exceed the physics update rate. As a result, the haptic feedback force field that the user feels, will never be inconsistent with his visual feedback. While this approach suffices for the purposes of physical travel, we are aware that this solution would not suffice for detailed feeling and recognition of object surfaces. However, this is not our purpose, and the approach taken suffices for the haptic travel technique.

Some issues were also raised by the forces concerning the terrain's slope factor in combination with triangular mesh representations. Since we use the normal of the terrain position at the character controller directly, discontinuities may be perceived as unpleasant force transitions. This can occur when a user is traveling fairly fast ($|movement_{XZ}| =$ large) and there is a hard slope transition (e.g. from downhill to uphill) resulting in a sudden change of force feedback in the Y-axis. To resolve this issue, we introduced a variable sized window of terrain normals. Instead of using the exact normal of the terrain, we now use an average of past normals. The calculation then becomes:

$$k_{sl} = \sum k_{sl}^{past} / windowsize. \tag{15.6}$$

This of course results in a small delay of the slope factor, but informal tests have shown that the window size does not have to be large (3 physical normals are sufficient) to get smooth transitions in all situations. Furthermore, since it takes several physical and visual frame updates to move the entire avatar over this transition zone, this delay is unnoticeable for users. By adapting the window size to the level of detail of the terrain, the application developer can always give the user smooth force transitions. When making the window smaller, there will be harder transitions, making it better for feeling the details of the terrain mesh. An illustration of the smoothing window is shown in Figure 15.3.

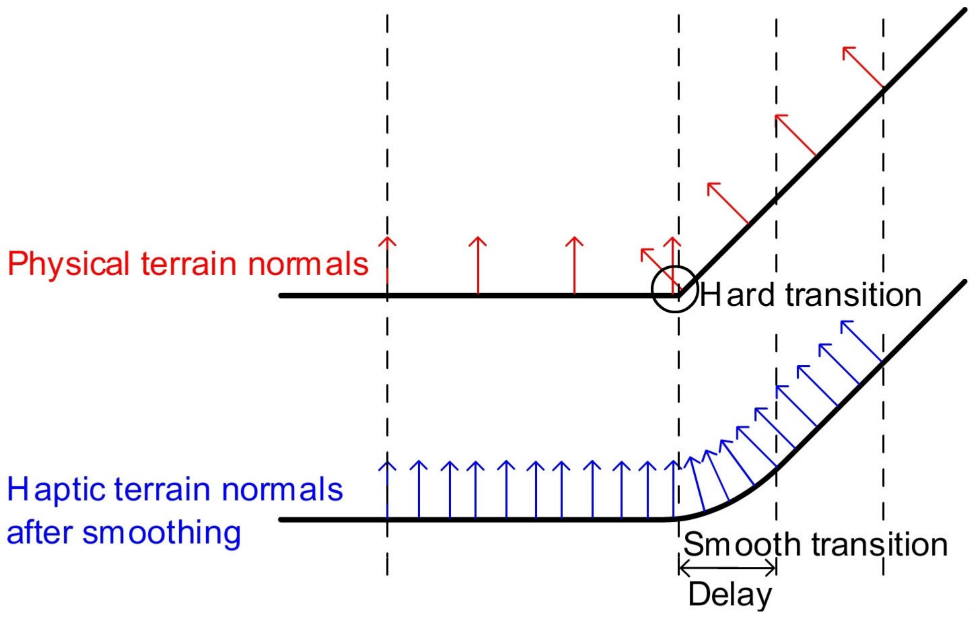


Figure 15.3: An illustration of the difference in the terrain normals and the haptic normals after applying the smoothing window. The delay is linearly dependent on the window size

CHAPTER 16

Evaluating the Effects of Haptics on Presence while Traveling

In the previous Chapter, we showed how we employed the rigid body simulator of the *ExtReAM* animation library in the haptic rendering loop to create dynamic haptic VEs. Since *navigation* is undoubtedly the most ubiquitous of the interaction tasks, we created a haptic travel technique allowing users to feel the terrain and surrounding dynamic objects while navigating through the VE. As VR tries to immerse the participant as much as possible, it is crucial that we investigate how the stimuli on the different senses influence the user's presence, the feeling of being there. Therefore, in this Chapter we present an experimental study on the influence of haptics on the perceived presence during haptic travel. More specifically, we investigate how adding haptic force feedback to travel affects: task performance, perceived task performance, perceived virtual presence, mental and physical workload. Furthermore, we investigate if this influence is larger on more experienced VR users than on novices.

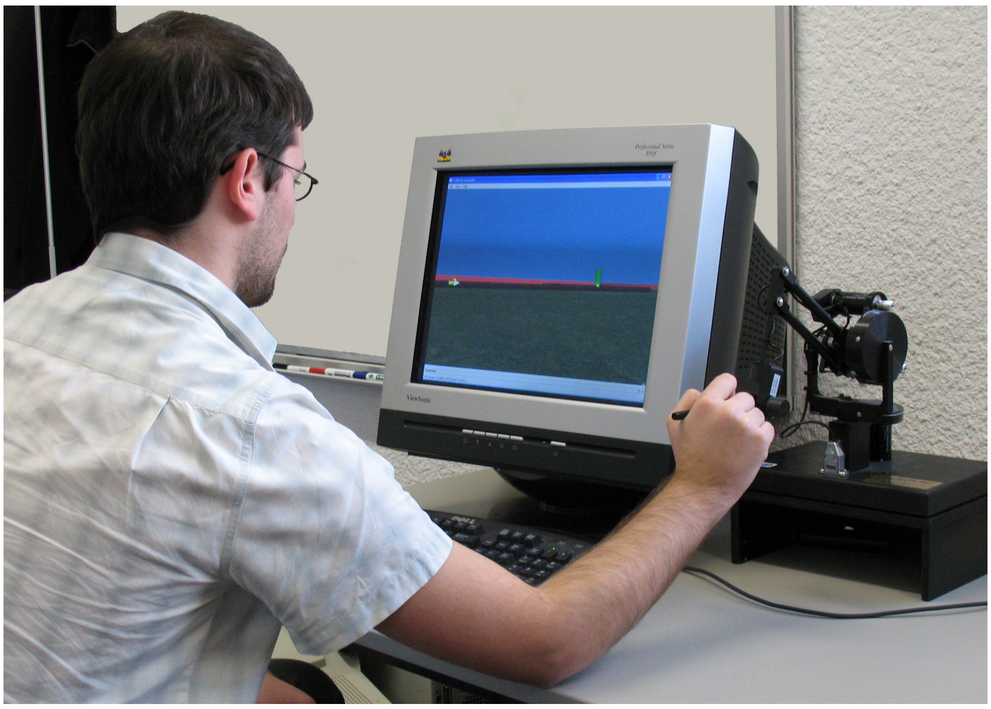


Figure 16.1: A user navigating in the desktop virtual environment using the haptic travel technique in first person camera mode.

16.1 The Haptic Travel Experiment

16.1.1 Virtual Environment Setup

The test environment that was used for the experiment, is based on the framework described in the previous parts. It consist of a desktop VE wherein 3D scenes and interactive objects can be loaded, and users can navigate. The *ExtReAM* library is used to increase realism, calculating the physical simulation and haptic force feedback, as explained in the previous Chapter. The desktop system employs a 19 inch monitor with a 120 Hz. refresh rate for the visual feedback. Audio is provided through a standard speaker set. The haptic display used in this investigation was a PHANToM premium 1.0 IO device which is placed as conveniently as possible at the user's dominant hand. Figure 16.1 shows a user who is using the PHANToM to navigate through the VE.

Three input methods were provided: one employing the keyboard and two using the PHANToM, based on the haptic travel of the previous Chapter. The keyboard method is used as a reference technique for the 3D input techniques, mainly in respect to task performance and workload. Since adding haptics to the keyboard is impossible, we cannot study the effect of that feedback on (perceived) presence. Another reason for using the keyboard as a reference is that, in contrast to the PHANToM, all participants are familiar with it, and it will show how efficient the 3D navigation techniques perform. For traveling, it uses only the arrow keys and provides: forward and backward movement at constant speed and left and right turns. In order to be able to objectively compare the PHANToM-based techniques to this, we had to adjust the PHANToM-based travel technique somewhat to make sure that all techniques provided the same functionality in terms of possible movements and speed. Therefore we disabled the left/right (strafe) movements, we switched to a constant turning speed and disabled the coupling of the traveling speed from the length of the displacement of the hand position from the neutral position. The final technique is exactly the same as the haptic travel technique, however the force feedback was left out. To summarize, we have 3 travel techniques allowing the same functionality: keyboard travel (KT), PHANToM travel without haptic feedback (PT) and PHANToM haptic travel (HT).

16.1.2 Task Description

In order to compare the different traveling techniques described above, a navigation task was set up. The users were asked to travel through several virtual scenes from a starting position toward an end position as fast as possible. For this task, ten different scenes were created, five having a flat terrain and five with sloped terrains. These scenes consist of a start position, a goal position and several physical barriers in the form of walls which, however, did not occlude the view of the goal position. The barriers were placed in order to coerce different travel patterns including e.g. moving in a straight line, moving through a narrow passage, and several combinations of short and long left and right turns. A schematic top down view of these scenes is shown in Figure 16.2. In order to eliminate the wayfinding component of the navigation task, large arrows were placed on the barriers showing the direction of travel at all times. The goal position was in all scenes represented as a high green cylinder. In order to let the users focus on the task at hand, there were no irrelevant objects placed in the scenes that could distract them or could lead to deviations of their travel path, e.g. if he would collide against a moving object.

Participants, when ready, could initiate the travel task by a simple button

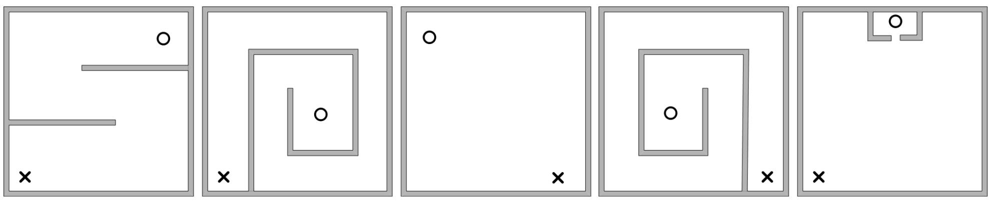


Figure 16.2: The scenes for the travel task, x marks the start position, o symbolizes the cylinder-shaped goal position which can be seen from the entire scene.

press. For keyboard navigation, the CTRL button was chosen, while in the case of the PHANToM techniques the PHANToM stylus button was used. Once started, the participant could travel freely to the goal position without interruption. Both the starting and finishing of a travel task were emphasized by playing a short sound. A flat scene with no obstacles was used for training. The user was free to move around until he felt reasonably comfortable in using the travel technique. Figure 16.3 illustrates a VE view of one of the participants while traveling towards the end goal in one of the virtual scenes.

16.1.3 Experimental Procedure

The design of the experiment is a within-participant design. The independent variables were the travel technique TT (KT, PT and HT) and the participant's VR experience PE (experienced (EX) or inexperienced (IX)). The dependent variables are trial completion time, travel distance, total rotations performed, perceived presence and workload.

The participants consist of unpaid volunteers that are screened based on questions regarding their experience with VEs and 3D input devices. On this basis a group of twelve was selected, nine male and three female, ranging in age from twenty-one to fifty-nine (average thirty-one). This includes six experienced and six inexperienced VR users. The experienced users consist of people that regularly used VR environments such as experienced 3D gamers and people that work with 3D input devices regularly. However, of these, only two users had prior experience with the PHANToM. During the experiment, participants use their dominant hand to control the input devices. Nine participants were right handed and three were left handed.

The travel techniques are fully counterbalanced across the twelve participants with one experienced and one inexperienced participant randomly assigned to each of the six unique orderings. The nine scenes, remaining after

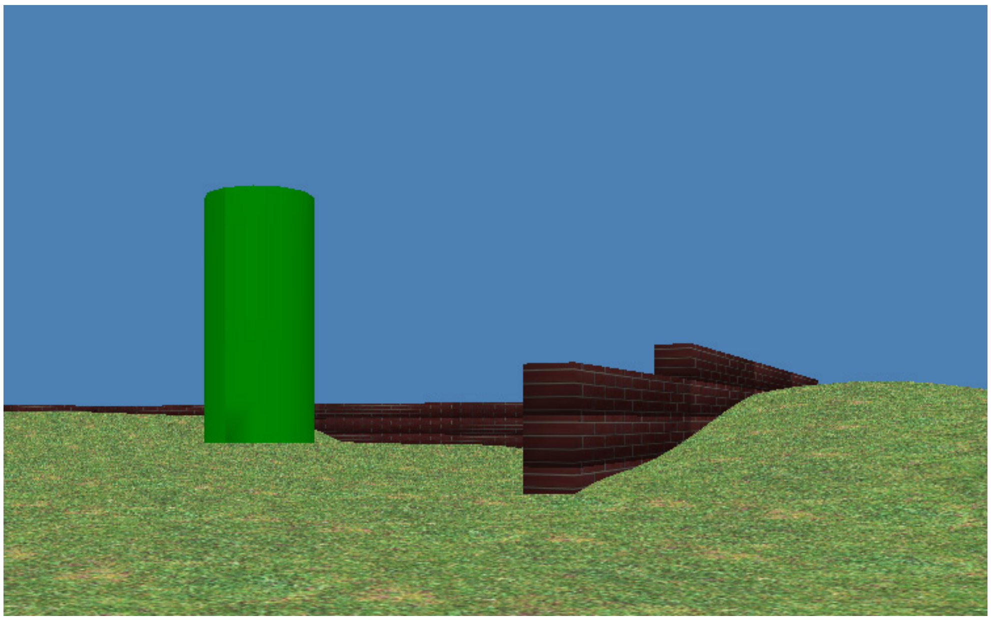


Figure 16.3: A user's view while traveling toward the end goal in the VE.

one was used for training, are presented in random order. After testing each travel technique the participant filled out the workload and perceived presence questionnaire. When the participant completes the entire test, another final questionnaire is presented in which he is asked to rank the travel techniques according to perceived performance and presence and can give other general remarks. Per participant, the experiment is performed in a single session, lasting about one hour.

16.1.4 Hypotheses

The main objective of this study is to test the influence of haptic feedback on the task of travel in VEs. More specifically we will check how the perceived presence, perceived performance, workload as well as task performance are influenced. Our main hypotheses include:

H1 Haptic force feedback improves task performance.

H2 Haptic force feedback increases perceived performance.

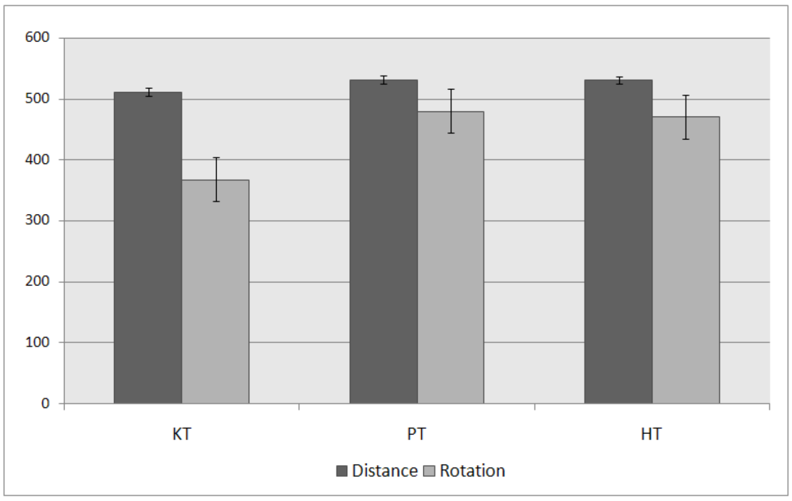
H3 Haptic force feedback increases perceived virtual presence.

H4 Haptic force feedback increases the physical demand.

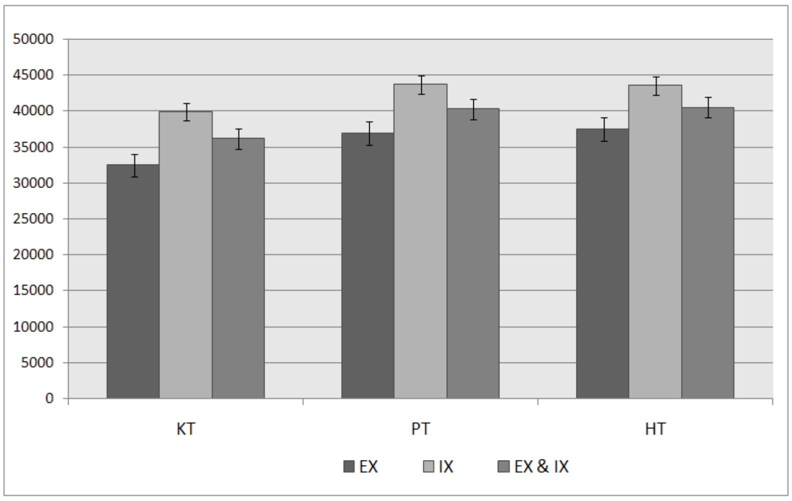
16.2 Experimental Results

16.2.1 Trial Completion Time

The trial completion time, is defined as the time it takes a participant to navigate from the start to the goal position. The average trial completion times were calculated per technique by averaging the trial completion times for all users and scenes. For the KT, this resulted in an average time of 36.2s, for PT this was 40.3s and the HT averaged 40.5s. Focusing on the PHANToM-based techniques, no statistical significant performance difference is present, refuting H1. Post hoc comparisons showed that keyboard was significantly faster than both PHANToM-based techniques ($p < .05$) in terms of timing, the increase is around 11%. In order to explain this difference, we first take a look at the other dependent variables: the distance traveled and the amount of rotations a user made. Statistical analysis showed that there is no significant difference in distance traveled between the different travel techniques, but as illustrated in figure 16.4(a), we can see that using the PHANToM results in slightly longer distances traveled (4%). As a result of the constant speed, this difference in traveled distance can already partly explain the longer trial completion times. A further explanation can be found by looking at the amount of degrees rotated during the navigation task. There is a significant difference between the keyboard technique and the PHANToM techniques ($p < .01$) which is also illustrated in figure 16.4(a). The (de)activation of the rotation with the PHANToM is slower compared to the keyboard since the user has to make a bigger gesture. For the PHANToM, the user had to rotate a certain amount before rotation was activated and in order to deactivate, the user had to rotate back to the neutral position which takes longer than pressing/releasing a key. As a result many users frequently overturned while traveling and others regularly halted in order to specify a new orientation before resuming forward movement again. This behavior only occurred very rarely with the KT, and only with IX, which might also explain the larger difference in completion time than in distance traveled. In the original haptic travel technique the rotation with the PHANToM would happen gradually according to the amount of rotation, which in this test was made constant in order to have a more fair comparison with the KT. We expect this problem of the tested technique to be less significant if we would have tested the original haptic travel technique, as it allows the rotation speed to be adjusted gradually.



(a)



(b)

Figure 16.4: (a) Total distance traveled and rotations performed by participants. (b) Trial completion time by participant's experience.

We found that there was a significant interaction effect with PE ($p < .0001$). By splitting up the users according to PE we see a significant difference in trail completion time. This interaction is illustrated in Figure 16.4(b). As expected, experienced participants performed faster than inexperienced participants but the performance with regard to the travel techniques has not been influenced significantly by participant's experience. We found similar results for the other dependent variables.

16.2.2 Workload

The participant's workload was estimated by using the NASA-TLX questionnaire [Hart 90]. Figure 16.5(a) shows the resulting scores (scored on a scale of 100). The KT scored lower on all categories, which logically results in a lower total workload of 21. The PHANToM-based techniques had a significantly higher total workload ($p < .01$), 40 for the PT and 39 for HT. When we considered the difference between EX and IX, we found there was a small difference between them. Experienced participants had a smaller total workload than inexperienced participants, which was expected.

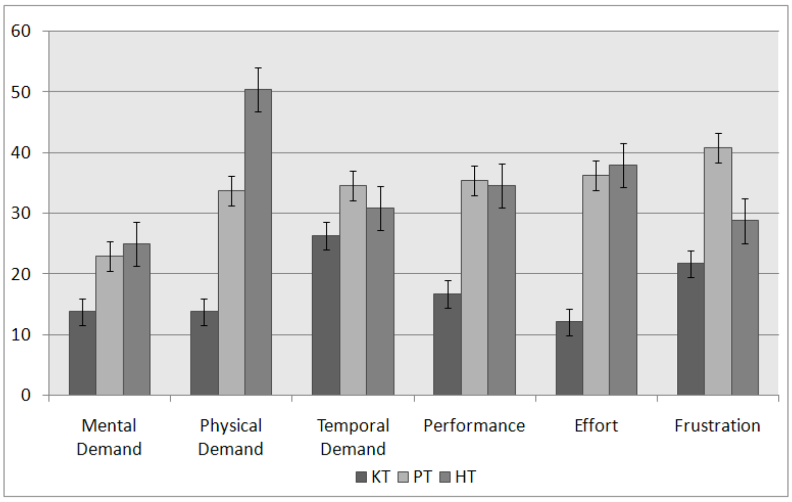
Looking into the different subscales, we found that there is no significant difference between PT and HT for mental demand, temporal demand, performance and effort. Physical demand, on the other hand, is substantially higher in the case of HT. This is an expected result, since the user had to constantly push against a varying force field in the direction he wanted to travel. This was also hypothesized as H4. In contrast, the frustration level significantly drops when haptic forces are present. We can at least partially attribute this to the fact that in case the user has feedback, that he can feel that he is moving. This is also present while navigating with the keyboard. The user just knows that he is moving when he presses a button (unless when he is colliding with an obstacle). When no haptics were present the user could only determine his movement visually. Also, in the PT, users made larger hand movements up to the device's workspace limits and experienced more difficulty in finding the neutral position. This concurs with the findings of Mine [Mine 95] and Bowman [Bowman 98] who state that many problems with 3D interaction techniques are caused by the lack of haptic feedback. Considering this in view of the total workload, we see that the decrease in frustration, caused by introducing feedback, is alleviated by an equally large increase in physical demand resulting in similar workloads for both techniques.

Figure 16.5(b) gives a more detailed overview of the workload for each travel technique taking into account participant's experience. When we compare EX with IX users, we can diagnose that there was an overall higher mental demand. This was to be expected, since users that are familiar with traveling in VEs need to concentrate less on the actions they perform. Both user groups also indicate an important increase in physical demand for the HT with respect to PT, the increase was larger for EX than for IX. Also, inexperienced users have a lower temporal demand than experienced users. This probably resulted from the constant speed condition, which was the same for all users. For experienced users the speed might not have been fast enough by which they felt more temporal demand, as they might have had the feeling they could have performed better (smaller trial completion times). During observations, we also saw, especially with EX, that when the users had reached the final straight line toward the end goal position, they enlarged their hand movement, trying to speed up. We consider this to be a confirmation of our choice in the original haptic travel technique, that allows users to define their virtual speed by the size of the displacement from the neutral position. If we look at performance, effort and frustration, there was a clear higher demand which can be attributed to them being less experienced.

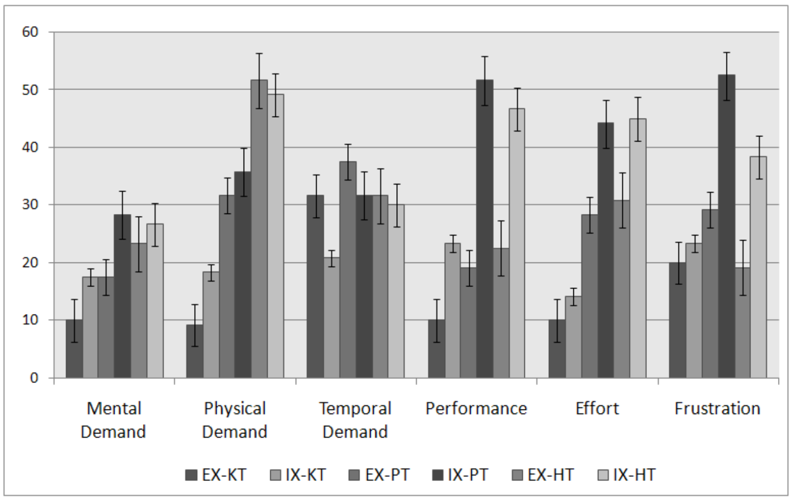
16.2.3 Perceived Presence and Performance

In order to estimate the perceived presence, the user had to fill out the Witmer and Singer presence questionnaire (version 4.0) [Witmer 98] and had to rank the travel techniques according the perceived presence and performance. This questionnaire is subdivided in 4 subscales: involvement, sensory fidelity, adaptation/immersion, interface quality. As this questionnaire is designed for total VR experiences, we could remove some questions that were irrelevant for our task (such as: "How closely were you able to examine objects?"), considering the fact that the environment and interaction possibilities remained the same. As a result, all questions regarding sensory fidelity seemed to be dismissed. Therefore, we will discuss only the remaining three.

Considering the PHANToM-based techniques Figure 16.6 shows that the addition of haptics gives a higher value for all 3 categories, especially for involvement. From this result, we can conduct that adding haptics to the travel technique gave the users a higher level of presence. Furthermore, these results are similar for both EX and IX. Comparing to the keyboard technique, the condition with the HT gives better results for involvement and adaptation/immersion but not for interface quality. This resulted mostly from the fact that users judged the interface quality on the basis of the task that was given, in which they were asked to perform the task as fast as possible. They felt this



(a)



(b)

Figure 16.5: (a) Workload by travel technique. (b) Workload by travel technique and participant's experience.

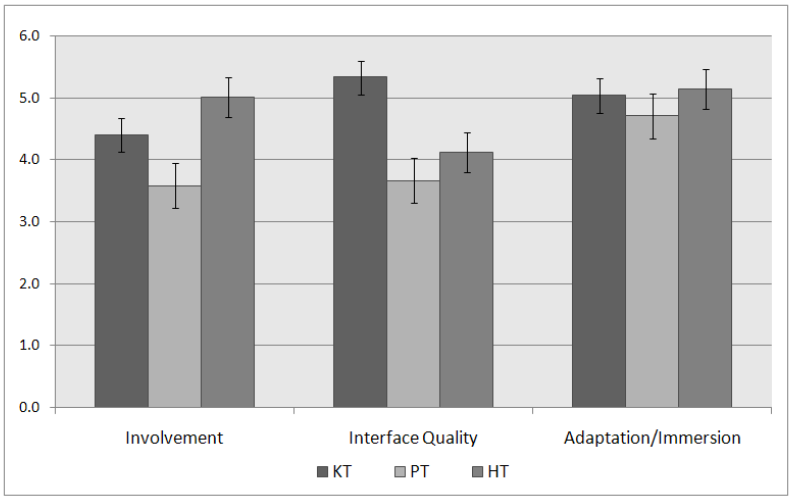


Figure 16.6: The 3 perceived presence categories by travel technique.

to be the KT, which is confirmed by their ranking of the navigation techniques according to perceived performance. All participants felt they were the fastest with the KT, which is sustained by the results described in section 16.2.1. The HT was ranked second and the PT last (χ^2 test: $p < .0001$) confirming H2. This is probably also caused by the fact that all users were acquainted with the keyboard, while only 2 of them had prior experience with the PHANToM device. The ranking order for perceived presence is the HT, which was preferred by 9 participants, KT by 2 participants and the PT was preferred by 1 participant (χ^2 test: $p < .01$). The participant that stated to perceive more presence with the technique of the PHANToM without haptics, was an experienced VR user who had several years of experience with 3D input devices including the PHANToM and the non-haptic MicroScribe-3D input device. This ordering of the TT by perceived presence, is confirmed by the results of the presence questionnaire and is an acknowledgement of H3.

CHAPTER 17

Conclusions

In contrast to the previous parts, where we focused on the realization of dynamic interactions and animations, here we focused more on the user aspects of interaction in the context of avatar control. We discussed how most contemporary VE systems only provide predefined animations to the user and how this results in unrealistic behavior and repetitiveness. We presented a general solution supporting on-the-fly creation of new animations employing inverse kinematics. As a demonstration, we created an interaction technique controlling a humanoid avatar's arm in a real-time NVE. The results dramatically increased the interaction possibilities of the user, as well as his identification with the controlled virtual representation. Furthermore, we explained how avatar control techniques can be distributed efficiently with minimal bandwidth consumption, making it more attractive for use in NVE systems.

We also discussed that most contemporary haptic applications are only provided in specialized setups with specialized collision detection algorithms due to high update requirements. As a solution, we presented a way to utilize a standard rigid body simulation system to provide information to calculate force feedback information. This was then used to create an interaction technique for haptic travel that allows a user to navigate through a virtual scene while using the PHANToM IO device for rendering force feedback on what is happening to his representation. Furthermore, we explained how we calculated the feedback from rigid body simulation, terrain, slope and collision information and discussed how we solved instabilities.

To determine the impact of our travel technique on users, we performed an empirical study on the addition of haptic force feedback to travel, and how this affects task performance, perceived task performance, perceived presence, mental and physical workload. A formal experiment was conducted comparing three travel techniques, a keyboard technique used as a benchmark and two PHANToM-based techniques, with and without haptic feedback. Our results demonstrate that, in terms of efficiency, the keyboard technique outperformed both PHANToM techniques, as expected. However, in terms of perceived user presence, involvement and satisfaction, haptic travel proved to be better. Comparison among the PHANTOM-based techniques showed that performance was not significantly improved although users perceived the haptic condition to be faster, allowing us to conclude that adding haptics has a positive influence on the task of travel. Also, we showed that these results apply to both experienced and inexperienced VR users.

Part V

CONCLUSIONS AND FUTURE RESEARCH

General Conclusions and Directions for Further Research

Throughout this dissertation, we presented our approaches to realize more dynamic virtual environments. This final part presents the main findings of our research. As each previous part already discussed some conclusions to the sub-topics, a more general approach is taken here, relating our solutions to the problem statement. In addition, we propose some directions for further research in this area.

General Conclusions

VEs and NVEs have evolved tremendously over the past decades and have become one of the most active areas of research within computer science. Although interactivity is one of the key terms in their definitions, this area has not been fully investigated yet. Most mechanisms supporting interaction consist of ad hoc solutions that are hard coded for specific applications and hardly any literature is found on the matter. Consequently, most contemporary systems support only limited interactive possibilities and dynamic behaviors. In this dissertation we took a bottom-up approach to realize a general solution to create more dynamic and interactive VEs.

The basic interactive object approach presents a general and dynamic interaction mechanism for VE systems. This was realized by employing the concepts of feature modeling to describe the interactive object properties. Furthermore, we generalized all objects that could be present in our interactive virtual worlds at the interaction level, so one central mechanism is able to handle all interactions. The major advantage of this approach is that all interaction information is located at object level and thus independent of the application. As a result, objects can change their appearance, structure and behaviors even at run-time and new objects, unknown to the application, can be inserted at any time. Also, as there is only one interaction mechanism which handles all objects, all the objects in a virtual world can interact with each other automatically. These properties allow for highly dynamic virtual worlds to be created. While modeling interactive virtual worlds using our approach might require more work in the modeling stage, interactive objects are highly reusable and depreciate the use of ad hoc interaction mechanisms.

Apart from the dynamics, another important aspect is the representation of actions occurring in virtual environments. For interactions to appear natural, realistic simulation and animations are required. This was provided by integrating *ExtReAM* into the interactive object approach. This library was devised as a platform-independent, easily extensible animation library. Plugins enable support for skeleton-based character animation techniques and rigid body simulation. This causes the virtual world objects to appear more physically correct and extends the interaction framework with new animation and physical simulation possibilities, resulting in much more natural and plausible virtual worlds.

Furthermore, the resulting realistic interactive and dynamic object approach supports dynamic avatar control which can be used to create more natural interaction techniques. By utilizing dynamic animation techniques we can overcome the unrealistic and repetitive results from standard approaches to interaction representation and give VE designers much more freedom. Also, we showed that rigid body simulation can be used to calculate feedback forces for the users. More dynamic interaction techniques and force feedback increase the user's feeling of identification with his virtual representation and raises the user's immersion.

Directions for Further Research

The research principles proposed in this dissertation are currently being extended to provide a general way to describe all entities in a (N)VE system, including objects that are not necessarily related to interaction. The approach taken is even more general in the way object properties are described. In this approach a scene is composed of a hierarchical tree of nodes. Objects are completely generalized and can be attached to these nodes. The objects consist of different components which describe their different aspects. Examples thereof include: graphical and physical representation, interaction information, audio information, etc. Furthermore, the system will utilize an efficient scripting engine (Lua), that can be used to control all aspects of the interactive virtual world. Currently we are also working on a modeler which allows the user to create interactive objects in a more graphical way. This will drastically improve the modeling of interactive objects and decrease the time necessary to create them.

With respect to *ExtReAM*, possible extensions include the creation of plugins that will enable other animation and simulation techniques such as controllable physically based animation and deformable objects. Also, the proposed interaction techniques can be extended and investigated further in several ways. First of all, we can employ the haptic force calculation of the haptic travel technique to add force feedback to the IK arm technique. This will resolve the problem we have with the user's hand moving further while the virtual hand is stopped by a collision. Also, both techniques could be combined resulting in complete haptic control over the embodiment and travel. Thirdly, the ITs should be compared to other interaction techniques, especially with respect to their influence on the user's presence and performance. Furthermore, studying the influence of the haptic travel on wayfinding could be interesting, also for visually impaired people.

Finally, efficient distribution of physical simulation among different clients on large scale networks with varying conditions is an active area of research. Integrating solutions for this problem in our approach will create the possibility to use it on a larger scale, increasing the applicability.

APPENDICES

APPENDIX A

Publications

[Quax 03] P. Quax, T. Jehaes, Jorissen & W. Lamotte. *A Multi-User Framework Supporting Video-Based Avatars.* In Proceedings of the 2nd workshop on Network and system support for games (NETGAMES), pp. 137 - 147, ACM Press, ISBN 1-58113-734-6, May 2003.

[Jorissen 04a] P. Jorissen & W. Lamotte. *A Framework Supporting General Object Interactions for Dynamic Virtual Worlds.* Proceedings of 4th International Symposium on Smart Graphics, pp. 154 - 158, ISBN 3-540-21977-3, ISSN 0302-9743, May 2004.

[Jorissen 04b] P. Jorissen, and M. Wijnants & W. Lamotte. *Using Collaborative Interactive Objects and Animation to Enable Dynamic Interactions in Collaborative Virtual Environments*, In Proceedings of CASA 2004, pp. 223 - 230, July 2004.

[Jorissen 05a] P. Jorissen, & W. Lamotte. *Dynamic Physical Interaction Platform for Collaborative Virtual Environments.* Proceedings of CollabTech 2005, pp. 79 - 84, ISBN 4-915256-57-X, July 2005.

[Jorissen 05b] P. Jorissen, M. Wijnants & W. Lamotte. *Dynamic Interactions in Physically Realistic Collaborative Virtual Environments.* In IEEE Transactions on Visualization and Computer Graphics, vol. 11, no. 6, pp.

649 - 660, ISSN 1077-2626, Nov. 2005.

[Jorissen 05c] P. Jorissen, J. Dierckx & W. Lamotte. *The ExtReAM Library: Extensible Real-time Animations for Multiple Platforms.* In Proceedings of the 7th International Conference on Computer Games (CGAMES). pp. 140 - 145, ISBN 0-9549016-2-6, Nov. 2005.

[Jorissen 06] P. Jorissen, J. De Boeck & W. Lamotte. *Bringing Haptics and Physical Simulation Together: Haptic Travel through Physical Worlds.* In Computer Animation and Virtual Worlds (CAVW), Special Issue CASA 2006, pp. 179 - 187, Wiley, ISSN 1546-4261, Aug. 2006.

[Di Fiore 07a] F. Di Fiore, P. Jorissen, F. Van Reeth, E. Lombaert, M. Valcke, G. Vansichem, P. Veevaete & L. Hauttekeet. *ASCIT sick children: Again at my School by fostering Communication through Interactive Technologies for long term sick children.* In Proceedings of the IASTED International Conference on Telehealth, pp. 102 - 107, IASTED and ACTA Press, ISBN 978-0-88986-667-6, CD ISBN: 978-0-88986-668-3, May 2007.

[Jorissen 07b] F. Di Fiore, P. Jorissen, G. Vansichem & F. Van Reeth. *A 3D Virtual Learning Environment to Foster Communication For Long Term Ill Children.* The 2nd International Conference of E-Learning and Games (Edutainment), pp. 92 - 103, Lecture Notes in Computer Science LNCS4469, Springer, ISSN 0302-9743, June 2007.

[Jorissen 07a] P. Jorissen, L. Vanacken, W. Lamotte & K. Coninx. *Evaluating the Effects of Haptics on Presence while Traveling in a Desktop Virtual Environment.* In Virtual Environments 2007 (IPT-EGVE) Short Papers and Posters, pp. 79 - 84, Eurographics Association, ISBN 978-3-905673-64-7, ISSN 1727-530X, Jul. 2007.

[Jorissen 07b] P. Jorissen, F. Di Fiore, G. Vansichem & W. Lamotte. *A Virtual Interactive Community Platform Supporting Education for Long-Term Sick Children.* International Conference on Cooperative Design, Visualization and Engineering (CDVE2007), Lecture Notes in Computer Science LNCS series, LNCS 4674, pp. 58 - 69, Springer, ISSN 0302-9743, Sep. 2007.

[Di Fiore 08] F. Di Fiore, P. Jorissen, F. Van Reeth, E. Lombaert, M. Valcke, G. Vansichem, P. Veevaete & L. Hauttekeet. *ASCIT sick children: Again at my School by fostering Communication through Interactive Technologies for long term sick children.* In Journal of Advanced Technology for Learning (ATL), vol. 5, no. 1, ACTA PRESS, Jan. 2008.

APPENDIX B

Dutch Summary - Samenvatting

De laatste decennia vormt het domein van de genetwerkte virtuele omgevingen een zeer actief onderzoeksgebied binnen de informatica. Samen met de groeiende capaciteiten van hedendaagse computers zorgen de dalende kosten voor hardware en connectiviteit ervoor dat deze technologie meer en meer beschikbaar wordt thuis en op de werkplek. Meerdere applicaties en specifieke prototypes worden momenteel dan ook in verschillende sectoren succesvol toegepast. Het overgrote deel van het gevoerde onderzoek en ontwikkeling binnen dit domein is echter gericht op de technologie, waardoor een van de belangrijkste componenten, namelijk interactie, veel minder aan bod komt.

In dit werk richten we ons op het realiseren van meer realistische virtuele ervaringen door de interactiekloof tussen de echte en de virtuele wereld te verkleinen. In tegenstelling tot ad hoc oplossingen proberen we een oplossing te bieden die algemeen en herbruikbaar is. De voorgestelde interactieve object aanpak voorziet zo een oplossing door een algemeen interactiemechanisme te toe te passen voor elke interactie binnen de virtuele wereld. Verder maakt deze aanpak gebruik van het feature modeling principe, dat objecten toelaat om hun eigen interactie-informatie te specificeren op objectniveau. Hierdoor kunnen interacties onafhankelijk van de applicatie worden voorgesteld, hetgeen ook toelaat om nieuwe objecten toe te voegen aan de applicatie tijdens de simulatie.

Het probleem van realistische interacties is echter niet beperkt to het representeren en uitvoeren van interacties, maar dient ook rekening te houden met andere domeinen zoals realistische simulatie en animatie, gebruikersrepresentatie en mens-computerinteractie. Daarom werd in een tweede fase van dit werk ExtReAM ontwikkeld en bijgevoegd. Deze platform-onafhankelijke animatie- en simulatiebibliotheek laat ons toe het realisme van de simulatie te verhogen en de interacties binnen de virtuele wereld realistischer voor te stellen. Verder wordt het ook mogelijk om objecten uit te rusten met fysieke eigenschappen en ze fysieke acties te laten uitvoeren. Door onze interactieve object aanpak te combineren met realistische simulatie kunnen we meer levendige virtuele werelden creëren met minimale inspanning, hetgeen bij correct gebruik zal leiden tot betere virtuele belevingen en hogere niveaus van ‘aanwezigheid’.

Meer natuurlijke interactie technieken kunnen ook het gevoel van onderdompeling in de virtuele wereld sterk verbeteren. Daarom onderzoeken we in dit werk ook hoe onze aanpak kan gebruikt worden om meer realistische controle over de gebruikersrepresentatie en natuurlijke interactietechnieken te realiseren. We presenteren daarom twee nieuwe 3D interactietechnieken die gerealiseerd werden met onze aanpak. De eerste techniek laat gebruikers toe om op een directe wijze met de virtuele wereld te interageren door hun virtuele hand te controleren met een 3D invoerapparaat. We bespreken hoe deze techniek tot stand kwam en leggen uit hoe ze gedistribueerd kan worden tussen de verschillende gebruikers, met een minimum aan bandbreedte gebruik. De tweede techniek voorziet een manier om te navigeren door de virtuele omgeving waarbij de gebruiker haptische feedback krijgt gebaseerd op wat er met zijn virtuele tegenhanger gebeurt. Dit werd gerealiseerd door rigid body simulatie informatie om te zetten naar krachtterugkoppeling. Met een formele gebruikersstudie tonen verder aan dat deze haptische navigatie methode het gevoel van onderdompeling in de virtuele wereld verhoogt.

APPENDIX C

Interactive Object Examples

C.1 Interactive Door Object

```
<?xml version="1.0" ?>
<!DOCTYPE CIOBJECT SYSTEM "ciobject.dtd">
<CIOBJECT object_name="slidingdoor">

  <OBJECT_PROPERTIES>
    <DESCRIPTION>
      This is a CIObject used for testing the CIFramework.
    </DESCRIPTION>

    <PART partid="doorframe" filename="glassdoorframe.ms3d" isfixed="TRUE">
      <POSITION x="0" y="3.8" z="0" />
      <ORIENTATION x="0" y="0" z="0" />
      <COLLISIONBOX
        xsize="8" ysize="0.2" zsize="0.5"
        xpos="0" ypos="0" zpos="0"
        xrot="0" yrot="0" zrot="0" />
    </PART>

    <PART partid="glassdoor_left" filename="glassdoor.ms3d" parentid="doorframe">
      <POSITION x="-2.9" y="1.85" z="0.1" />
      <ORIENTATION x="0" y="0" z="0" />
      <COLLISIONBOX
        xsize="2" ysize="3.8" zsize="0.1"
        xpos="0" ypos="0" zpos="0"
        xrot="0" yrot="0" zrot="0" />
      <PART_CONSTRAINTS>
        <MAX_TRANSLATE_CONSTRAINT upx="2.0" upy="0" upz="0"
                                  lowerx="0" lowery="0" lowerz="0" />
        <MAX_ROTATE_CONSTRAINT clockwisex="0" clockwisey="0" clockwisez="0"
                               cclockwisex="0" cclockwisey="0" cclockwisez="0" />
      </PART_CONSTRAINTS>
    </PART>

    <PART partid="glassdoor_right" filename="glassdoor.ms3d" parentid="doorframe">
      <POSITION x="2.9" y="1.85" z="0.1" />
      <ORIENTATION x="0" y="0" z="0" />
      <COLLISIONBOX
        xsize="2" ysize="3.8" zsize="0.1"
        xpos="0" ypos="0" zpos="0"
        xrot="0" yrot="0" zrot="0" />
      <PART_CONSTRAINTS>
        <MAX_TRANSLATE_CONSTRAINT upx="0" upy="0" upz="0"
                                  lowerx="-2.0" lowery="0" lowerz="0" />
        <MAX_ROTATE_CONSTRAINT clockwisex="0" clockwisey="0" clockwisez="0"
                               cclockwisex="0" cclockwisey="0" cclockwisez="0" />
      </PART_CONSTRAINTS>
    </PART>

    <PART partid="fixedglass_left" filename="glassdoor.ms3d" parentid="doorframe" isfixed="TRUE">
      <POSITION x="3" y="1.85" z="0" />
      <ORIENTATION x="0" y="0" z="0" />
      <COLLISIONBOX
        xsize="2" ysize="3.8" zsize="0.1"
        xpos="0" ypos="0" zpos="0"
        xrot="0" yrot="0" zrot="0" />
    </PART>

    <PART partid="fixedglass_right" filename="glassdoor.ms3d" parentid="doorframe" isfixed="TRUE">
      <POSITION x="-3" y="1.85" z="0" />
      <ORIENTATION x="0" y="0" z="0" />
      <COLLISIONBOX
        xsize="2" ysize="3.8" zsize="0.1"
        xpos="0" ypos="0" zpos="0"
        xrot="0" yrot="0" zrot="0" />
    </PART>

    <PART partid="buttonbase1" filename="buttonbase.ms3d" parentid="doorframe" isfixed="TRUE">
      <POSITION x="2.5" y="0.8" z="1.0" />
      <ORIENTATION x="0" y="0" z="0" />
      <PHYSICSCOLLISIONBOX
        xsize="0.3" ysize="1.4" zsize="0.3"
        xpos="0" ypos="0" zpos="0"
        xrot="0" yrot="0" zrot="0" />
    </PART>

    <PART partid="button1" filename="buttonsphere.ms3d" parentid="buttonbase1" isfixed="TRUE">
      <POSITION x="2.5" y="1.2" z="1.2" />
      <ORIENTATION x="0" y="0" z="0" />
      <COLLISIONBOX
        xsize="0.2" ysize="0.2" zsize="0.2"
        xpos="0" ypos="0" zpos="0"
        xrot="0" yrot="0" zrot="0" />
    </PART>

    <PART partid="buttonbase2" filename="buttonbase.ms3d" parentid="doorframe" isfixed="TRUE">
      <POSITION x="2.5" y="0.8" z="-1" />
      <ORIENTATION x="0" y="0" z="0" />
      <COLLISIONBOX
        xsize="0.3" ysize="1.4" zsize="0.3"
        xpos="0" ypos="0" zpos="0"
        xrot="0" yrot="0" zrot="0" />
    </PART>


    <PART partid="button2" filename="buttonsphere.ms3d" parentid="buttonbase2" isfixed="TRUE">
      <POSITION x="2.5" y="1.2" z="-1.2" />
      <ORIENTATION x="0" y="0" z="0" />
      <COLLISIONBOX
        xsize="0.2" ysize="0.2" zsize="0.2"
        xpos="0" ypos="0" zpos="0"
        xrot="0" yrot="0" zrot="0" />
    </PART>

    <ACTION name="open_left_door">
      <TRANSLATE_PART partid="glassdoor_left" x="-2" y="0" z="0" time="2000" />
```

```
    </ACTION>

    <ACTION name="open_right_door">
      <TRANSLATE_PART partid="glassdoor_right" x="2" y="0" z="0" time="2000" />
    </ACTION>

    <ACTION name="close_left_door">
      <TRANSLATE_PART partid="glassdoor_left" x="2" y="0" z="0" time="2000" />
    </ACTION>

    <ACTION name="close_right_door">
      <TRANSLATE_PART partid="glassdoor_right" x="-2" y="0" z="0" time="2000" />
    </ACTION>

    <VARIABLE type="bool" name="isClosed" value="TRUE" />
    <VARIABLE type="bool" name="doorsMoving" value="FALSE" />

  </OBJECT_PROPERTIES>

  <INTERACTION_PROPERTIES>

    <OBJECT_COMMAND command="MoveDoors" />

    <INTERACTION_ZONE zone_name="button1zone">
      <PART_REGION regionid="frontbuttonregion" partid="button1" />
    </INTERACTION_ZONE>

    <INTERACTION_ZONE zone_name="button2zone">
      <PART_REGION regionid="backbuttonregion" partid="button2" />
    </INTERACTION_ZONE>

    <TRIGGERS>
      <ZONETRIGGER triggerid="door_trigger" zones="button1zone,␣button2zone" />
    </TRIGGERS>

  </INTERACTION_PROPERTIES>

  <OBJECT_BEHAVIORS>
    <SCRIPT name="MoveDoorsScript" maxsimultaneousexecutions="1" maxexecutions="unlimited">
        push_object_var doorsMoving     <!-- if doors are already moving stop -->
        jump_if_false 3
        end
        set_object_var doorsMoving 1    <!-- else set doorsMoving true -->
        push_object_var isClosed        <!-- if doors are closed goto open commands -->
        jump_if_true 13
        pusharg_const close_left_door   <!-- call actions to close both doors -->
        call_command performAction
        pusharg_const close_right_door
        call_command performAction
        set_object_var isClosed 1       <!-- set doorsClosed variable true -->
        set_object_var doorsMoving 0    <!-- set doorsMoving variable false -->
        end                             <!-- end -->

        set_object_var doorsMoving 1    <!-- similar for opening -->
        pusharg_const open_left_door
        call_command performAction
        pusharg_const open_right_door
        call_command performAction
        set_object_var isClosed 0
        set_object_var doorsMoving 0
        end
    </SCRIPT>

    <TRIGGERCOMMAND triggerid="door_trigger" commandname="MoveDoors" />

    <COMMANDSCRIPT commandname="MoveDoors" scriptname="MoveDoorsScript" />

  </OBJECT_BEHAVIORS>

</CIOBJECT>
```

C.2 Physical Interactive Door Object

```
<?xml version="1.0" ?>
<!DOCTYPE CIOBJECT SYSTEM "ciobject.dtd">
<CIOBJECT object_name="glassdoor">

  <OBJECT_PROPERTIES>
    <DESCRIPTION>
      This is the interactive object description for a button controlled sliding door.
      The door can be opened and closed by collision triggers coupled to the buttons.
    </DESCRIPTION>

    <PART partid="doorframe" filename="glassdoorframe.ms3d" isfixed="TRUE">
      <POSITION x="0.0" y="3.8" z="0.0" />
      <ORIENTATION x="0.0" y="0.0" z="0.0" />
      <PHYSICSCOLLISIONBOX
        xsize="8" ysize="0.2" zsize="0.5"
        xpos="0.0" ypos="0" zpos="0"
        xrot="0.0" yrot="0.0" zrot="0.0" />
    </PART>

    <PART partid="glassdoor_left" filename="glassdoor.ms3d" parentid="doorframe">
      <POSITION x="-2.9" y="1.85" z="0.1" />
      <ORIENTATION x="0.0" y="0.0" z="0.0" />
      <PHYSICSCOLLISIONBOX
      xsize="2" ysize="3.8" zsize="0.1"
      xpos="0.0" ypos="0" zpos="0"
      xrot="0.0" yrot="0.0" zrot="0.0" materialname="unbreakableglass" />
      <PART_CONSTRAINTS>
        <MAX_TRANSLATE_CONSTRAINT upx="2.0" upy="0.0" upz="0.0"
                                  lowerx="0.0" lowery="0.0" lowerz="0.0" />
        <MAX_ROTATE_CONSTRAINT clockwisex="0.0" clockwisey="0.0" clockwisez="0.0"
                               cclockwisex="0.0" cclockwisey="0.0" cclockwisez="0.0" />
      </PART_CONSTRAINTS>
    </PART>

    <PART partid="glassdoor_right" filename="glassdoor.ms3d" parentid="doorframe">
      <POSITION x="2.9" y="1.85" z="0.1" />
      <ORIENTATION x="0.0" y="0.0" z="0.0" />
      <PHYSICSCOLLISIONBOX
      xsize="2" ysize="3.8" zsize="0.1"
      xpos="0.0" ypos="0" zpos="0"
      xrot="0.0" yrot="0.0" zrot="0.0" materialindex="1" />
      <PART_CONSTRAINTS>
        <MAX_TRANSLATE_CONSTRAINT upx="0.0" upy="0.0" upz="0.0" lowerx="-2.0"
                                  lowery="0.0" lowerz="0.0" />
        <MAX_ROTATE_CONSTRAINT clockwisex="0.0" clockwisey="0.0" clockwisez="0.0"
                               cclockwisex="0.0" cclockwisey="0.0" cclockwisez="0.0"/>
      </PART_CONSTRAINTS>
    </PART>

    <PART partid="fixedglass_left" filename="glassdoor.ms3d" parentid="doorframe" isfixed="TRUE">
      <POSITION x="3" y="1.85" z="0.0" />
      <ORIENTATION x="0.0" y="0.0" z="0.0" />
      <PHYSICSCOLLISIONBOX
      xsize="2" ysize="3.8" zsize="0.1"
      xpos="0.0" ypos="0" zpos="0"
      xrot="0.0" yrot="0.0" zrot="0.0" />
    </PART>

    <PART partid="fixedglass_right" filename="glassdoor.ms3d" parentid="doorframe" isfixed="TRUE">
      <POSITION x="-3" y="1.85" z="0.0" />
      <ORIENTATION x="0.0" y="0.0" z="0.0" />
      <PHYSICSCOLLISIONBOX
      xsize="2" ysize="3.8" zsize="0.1"
      xpos="0.0" ypos="0" zpos="0"
      xrot="0.0" yrot="0.0" zrot="0.0" />
    </PART>

    <PART partid="buttonbase1" filename="buttonbase.ms3d" parentid="doorframe" isfixed="TRUE">
      <POSITION x="2.5" y="0.8" z="1.0" />
      <ORIENTATION x="0.0" y="0.0" z="0.0" />
      <PHYSICSCOLLISIONBOX
      xsize="0.3" ysize="1.4" zsize="0.3"
      xpos="0.0" ypos="0.0" zpos="0.0"
      xrot="0.0" yrot="0.0" zrot="0.0" />
    </PART>

    <PART partid="buttonfront" filename="buttonsphere.ms3d" parentid="buttonbase1" isfixed="TRUE">
      <POSITION x="2.5" y="1.2" z="1.2" />
      <ORIENTATION x="0.0" y="0.0" z="0.0" />
      <PHYSICSCOLLISIONBOX
      xsize="0.2" ysize="0.2" zsize="0.2"
      xpos="0.0" ypos="0.0" zpos="0.0"
      xrot="0.0" yrot="0.0" zrot="0.0" materialname="steel" />
    </PART>

    <PART partid="buttonbase2" filename="buttonbase.ms3d" parentid="doorframe" isfixed="TRUE">
      <POSITION x="2.5" y="0.8" z="-1" />
      <ORIENTATION x="0.0" y="0.0" z="0.0" />
      <PHYSICSCOLLISIONBOX
      xsize="0.3" ysize="1.4" zsize="0.3"
      xpos="0.0" ypos="0.0" zpos="0.0"
      xrot="0.0" yrot="0.0" zrot="0.0" />
    </PART>

    <PART partid="buttonback" filename="buttonsphere.ms3d" parentid="buttonbase2" isfixed="TRUE">
      <POSITION x="2.5" y="1.2" z="-1.2" />
      <ORIENTATION x="0.0" y="0.0" z="0.0" />
      <PHYSICSCOLLISIONBOX
      xsize="0.2" ysize="0.2" zsize="0.2"
      xpos="0.0" ypos="0.0" zpos="0.0"
      xrot="0.0" yrot="0.0" zrot="0.0" />
    </PART>

    <JOINTS>
      <PRISMATICJOINT jointname="sliderright" partid="0" partid2="glassdoor_right"
          xorientation="1" yorientation="0" zorientation="0"
          xleftextent="2" xrightextent="0" />

      <PRISMATICJOINT jointname="sliderleft" partid="fixedglass_left" partid2="glassdoor_left"
          xorientation="1" yorientation="0" zorientation="0"
          xleftextent="0" xrightextent="2" />
    </JOINTS>


    <ACTION name="open_left_door">
      <TRANSLATE_PART partid="glassdoor_left" x="-2" y="0.0" z="0" time="2000" />
    </ACTION>
    <ACTION name="open_right_door">
      <TRANSLATE_PART partid="glassdoor_right" x="2" y="0.0" z="0" time="2000" />
    </ACTION>
    <ACTION name="close_left_door">
      <TRANSLATE_PART partid="glassdoor_left" x="2" y="0.0" z="0" time="2000" />
    </ACTION>
    <ACTION name="close_right_door">
      <TRANSLATE_PART partid="glassdoor_right" x="-2" y="0.0" z="0" time="2000" />
    </ACTION>
    <ACTION name="open_left_velocity">
      <SET_PART_LINEARVELOCITY partid="glassdoor_left" xvel="-5.3" yvel="0.0" zvel="0" time="2000" />
    </ACTION>
    <ACTION name="close_left_velocity">
      <SET_PART_LINEARVELOCITY partid="glassdoor_left" xvel="5.3" yvel="0.0" zvel="0" time="2000" />
    </ACTION>
    <ACTION name="open_right_velocity">
      <SET_PART_LINEARVELOCITY partid="glassdoor_right" xvel="5.3" yvel="0.0" zvel="0" time="2000" />
    </ACTION>
    <ACTION name="close_right_velocity">
      <SET_PART_LINEARVELOCITY partid="glassdoor_right" xvel="-5.3" yvel="0.0" zvel="0" time="2000" />
    </ACTION>

    <VARIABLE type="BOOL" name="isClosed" value="false" />
    <VARIABLE type="BOOL" name="doorsMoving" value="false" />

  </OBJECT_PROPERTIES>

  <INTERACTION_PROPERTIES>
    <OBJECT_COMMAND command="MoveDoors" />
```

```
    <INTERACTION_ZONE zone_name="button1zone">
      <PART_REGION regionid="frontbuttonregion" partid="button1" />
    </INTERACTION_ZONE>

    <INTERACTION_ZONE zone_name="button2zone">
      <PART_REGION regionid="backbuttonregion" partid="button2" />
    </INTERACTION_ZONE>

    <TRIGGERS>
      <ZONETRIGGER triggerid="door_trigger" zones="button1zone,␣button2zone" />
    </TRIGGERS>
  </INTERACTION_PROPERTIES>

  <OBJECT_BEHAVIORS>

    <SCRIPT name="door_script_physical" maxsimultaneousexecutions="1" maxexecutions="unlimited">
        push_object_var doorsMoving
        jump_if_false 3
        end
        set_object_var doorsMoving 1
        push_object_var isClosed
        jump_if_true 14
        pusharg_const close_left_doorphysical
        call_command performAction
        pusharg_const close_right_doorphysical
        call_command performAction
        set_object_var isClosed 1
        wait 700
        set_object_var doorsMoving 0
        end
        set_object_var doorsMoving 1
        pusharg_const open_left_doorphysical
        call_command performAction
        pusharg_const open_right_doorphysical
        call_command performAction
        set_object_var isClosed 0
        wait 700
        set_object_var doorsMoving 0
        end
//end of script
    </SCRIPT>

    <TRIGGERCOMMAND triggerid="door_trigger" commandname="MoveDoors" />
    <COMMANDSCRIPT commandname="MoveDoors" scriptname="door_script_physical" />

  </OBJECT_BEHAVIORS>
</CIOBJECT>
```

REFERENCES

[3ds Max 07] Autodesk 3ds Max. *http://usa.autodesk.com/*. World Wide Web, 2007.

[Anderson 98] T. Anderson. Flight: An advanced human-computer interface and application development environment. Master's thesis, University of Washington, 1998.

[Anderson 01] E. Anderson. *Real-Time Character Animation for Computer Games*. National Centre for Computer Animation, Bournemouth University, 2001.

[APA 07] American Psychological Association APA. *Dictionary.com Unabridged (v 1.1) http://www.dictionary.com*. World Wide Web, 2007.

[Badawi 06] M. Badawi. *Synoptic Objects Describing Generic Interaction Processes to Autonomous Agents in an Informed Virtual environment*. PhD thesis, Institut National des Sciences Appliées (INSA), 2006.

[Belleman 03] R. Belleman. *Interactive Exploration in Virtual Environments*. PhD thesis, Universiteit van Amsterdam, 2003.

[Benford 95] S. Benford, J. Bowers, L. Fahlén, C. Greenhalgh & D. Snowdon. *User embodiment in collaborative virtual environments*. In Proceedings of the SIGCHI conference on Human factors in computing systems, pages 242–249. ACM Press/Addison-Wesley Publishing Co., 1995.

[Berkley 03] J. Berkley. *Haptic Devices.* Rapport technique White Paper, Mimic Technologies, Seattle, May 2003.

[Bidarra 99] A.R.E. Bidarra. *Validity Maintenance in Semantic Feature Modeling.* PhD thesis, Technische Universiteit Delft, 1999.

[Bierz 05] T. Bierz, P. Dannenmann, K. Hergenröther, M. Bertram, H. Barthel, G. Scheuermann & H. Hagen. *Getting in Touch with a Cognitive Character.* In Proceedings of Worldhaptics '05, pages 440–445, march 2005.

[Blender 07] Blender. *http://www.blender.org/.* World Wide Web, 2007.

[Boulanger 06] J. Boulanger, J. Kienzle & C. Verbrugge. *Comparing interest management algorithms for massively multiplayer games.* In NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games, New York, NY, USA, 2006. ACM.

[Bowman 97] D. Bowman, D. Koller & L. Hodges. *Travel in Immersive Virtual Environments: An Evaluation of Viewpoint Motion Control Techniques.* In VRAIS '97: Proceedings of the 1997 Virtual Reality Annual International Symposium, pages 45–52, Albuquerque, NM, USA, march 1–3 1997. IEEE Computer Society.

[Bowman 98] D. Bowman. *Interaction Techniques for Immersive Virtual Environments: Design, Evaluation, and Application.* In Boaster paper presented at the Human-Computer Interaction Consortium (HCIC) Conference, 1998.

[Bowman 99] D. Bowman. *Interaction Techniques for Common Tasks in Immersive Virtual Environments: Design, Evaluation and Application.* PhD thesis, Georgia Institute of Technology, 1999.

[Bricken 94] W. Bricken & G. Coco. *The VEOS Project.* Presence, vol. 3, no. 2, pages 111–129, 1994.

[Broll 95] W. Broll. *Interacting in Distributed Collaborative Virtual Environments.* In Proceedings of the IEEE Virtual Reality International Symposium, pages 148–155, Los

Almitos, 1995.

[Cal3D 07] Cal3D. *http://cal3d.sourceforge.net/*. World Wide Web, 2007.

[Capin 98] T. Capin, Pandzic I., Thalmann D. & Thalmann N. *Realistic Avatars and Autonomous Virtual Humans in VLNET Networked Virtual Environments*. IEEE Virtual Worlds on the Internet, pages 157–173, 1998.

[Capps 00] M. Capps, D. McGregor, D. Brutzman & M. Zyda. *NPSNET-V: A New Beginning for Dynamically Extensible Virtual Environments*. IEEE Computer Graphics and Applications, vol. 20, no. 5, pages 12–15, 2000.

[Casalta 99] D. Casalta, Y Guiard & M. Lafon. *Evaluating two-handed input techniques: rectangle editing and navigation*. In CHI '99 extended abstracts on Human factors in computing systems, pages 236–237, New York, NY, USA, 1999. ACM.

[Casanueva 01] J. Casanueva & E. Blake. *The Effects of Avatars on Co-presence in a Collaborative Virtual Environment*. Technical Report CS01-02-00, Department of Computer Science, University of Cape Town, South Africa, 2001.

[Chadwick 89] J. Chadwick, Haumann D. & R. Parent. *Layered construction for deformable animated characters*. In SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques, pages 243–252, New York, NY, USA, 1989. ACM Press.

[Chen 06] I. Chen. *Olfactory Display: Development and Application in Virtual Reality Therapy*. ICAT, pages 580–584, 2006.

[Davide 01] F. Davide, M. Holmberg & I. Lundstrm. Virtual olfactory interfaces: electronic noses and olfactory displays, chapitre 12, pages 193–219. IOS Press, Amsterdam, 2001.

[De Boeck 02] J. De Boeck & K. Coninx. *Haptic Camera Manipulation: Extending the Camera-in-Hand Metaphor*. In Proceedings of Eurohaptics 2002, pages 36–40, Edinburgh, UK, 2002.

[De Boeck 05] J. De Boeck, C. Raymaekers & K. Coninx. *Are Existing Metaphors in Virtual Environments Suitable for*

Haptic Interaction. In Proceedings of 7th International Conference on Virtual Reality (VRIC 2005), pages 261–268, Laval, France, Apr. 2005.

[Dommel 97] H. Dommel & J. Garcia-Luna-Aceves. *Floor control for multimedia conferencing and collaboration*. Multimedia Systems, vol. 5, no. 1, pages 23–38, 1997.

[Dynamics 07] Newton Game Dynamics. *http://www.physicsengine.com/*. World Wide Web, 2007.

[Endorphin 07] NaturalMotion Endorphin. *http://www.naturalmotion.com/*. World Wide Web, 2007.

[Faloutsos 01a] P. Faloutsos, M. van de Panne & D. Terzopoulos. *Composable controllers for physics-based character animation*. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH '01), pages 251–260, New York, NY, USA, 2001. ACM Press.

[Faloutsos 01b] P. Faloutsos, M. van de Panne & D. Terzopoulos. *Composable controllers for physics-based character animation*. In SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pages 251–260, New York, NY, USA, 2001. ACM Press.

[Fisch 03] A. Fisch, C. Mavroidis, J. Melli-Huber & Y. Bar-Cohen. Haptic devices for virtual reality, telepresence, and human-assistive robotics, chapitre 4. SPIE Press, 2003.

[Fraser 00] M. Fraser. *Working with objects in collaborative virtual environments*. PhD thesis, University of Nottingham, 2000.

[Funkhouser 95] T Funkhouser. *RING: A Client-Server System for Multi-User Virtual Environments*. In Proceedings of the 1995 symposium on Interactive 3D graphics, pages 85–92. ACM Press, 1995.

[Giang 00] T. Giang, R. Mooney, C. Peters & C. O'Sullivan. *Real-Time Character Animation Techniques*. Rapport technique TCD-CS-2000-06, Image Synthesis Group Trinity College Dublin, Feb. 2000.

[Glencross 05] M. Glencross, M. Otaduy & A. Chalmers. *Interaction in distributed virtual environments*. Eurographics Tutorial, Aug. 2005.

[Granny3D 07] Granny3D. *http://www.radgametools.com/gramain.htm*. World Wide Web, 2007.

[Greenhalgh 95] C. Greenhalgh & S. Benford. *MASSIVE: a collaborative virtual environment for teleconferencing*. ACM Transactions on Computer-Human Interaction, vol. 2, no. 3, pages 239–261, 1995.

[Greenhalgh 96] C. Greenhalgh. *Dynamic, embodied multicast groups in MASSIVE-2*. Rapport technique NOTTCS-TR-96-8, Department of Computer Science,The University of Nottingham, Dec 1996.

[Hagsand 96] O. Hagsand. *Interactive Multiuser VEs in the DIVE System*. IEEE MultiMedia, vol. 3, no. 1, pages 30–39, 1996.

[Half-life 2 07] Half-life 2. *http://www.half-life2.com/*. World Wide Web, 2007.

[Hand 97] C. Hand. *A Survey of 3D Interaction Techniques*. Computer Graphics Forum 16(5), vol. 16, no. 5, pages 269–281, 1997.

[Hart 90] S. Hart & C Wickens. *Workload assessment and prediction*. In MANPRINT, An approach to systems integration, pages 257–296, 1990.

[Havok 07] Havok. *http://www.havok.com/*. World Wide Web, 2007.

[Hendrix 96] C. Hendrix & B. Woodrow. *The Sense of Presence Within Auditory Virtual Environments*. Presence: Teleoperators and Virtual Environments, vol. 5, no. 3, pages 290–301, 1996.

[Hinckley 94] K. Hinckley, R. Pausch, J. Goble & N. Kassell. *Moving Objects in Space: Exploiting Proprioception in Virtual-Environment Interaction*. In Proceedings of the 7th annual ACM symposium on User interface software and technology, 1994.

[Hinckley 98] K. Hinckley, M. Czerwinski & M. Sinclair. *Interaction and modeling techniques for desktop two-handed input*. In UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology, pages 49–58, New York, NY, USA, 1998. ACM.

[Insko 03] B. Insko. *Measuring presence: Subjective, behavioral and physiological methods*. In Being There: Concepts, effects and measurement of user presence in synthetic environments, pages 110–118, Amsterdam, NL, 2003. Ios Press.

[Jehaes 05] T. Jehaes, P. Quax & W. Lamotte. *Adapting a large scale networked virtual environment for display on a PDA.* In Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology (ACE '05), pages 217–220, New York, NY, USA, 2005. ACM Press.

[Joslin 04] C. Joslin, T. Di Giacomo & N. Magnenat-Thalmann. *Collaborative virtual environments: from birth to standardization.* IEEE Communications Magazine, vol. 42, no. 4, pages 28–33, Apr. 2004.

[Kallmann 98] M. Kallmann & D. Thalmann. *Modeling Objects for Interactive Tasks.* In Proceedings of the 9th Eurographics Workshop on Animation and Simulation (EGCAS'98), Lisbon, 1998.

[Kallmann 01] M. Kallmann. *Object Interaction in Real-Time Virtual Environments.* PhD thesis, École Polytechnique Fédérale de Lausanne (EPFL), 2001.

[Kovar 02] L. Kovar, M. Gleicher & F. Pighin. *Motion graphs.* In Proceedings of ACM SIGGRAPH 02, pages 473–482, 2002.

[Lander 98] J. Lander. *Skin Them Bones: Game Programming for the Web Generation.* In Game Developer Magazine, pages 11–16, May 1998.

[Levinson 96] L. Levinson. *Connecting Planning and Acting: Towards an Architecture for Object-Specific Reasoning.* PhD thesis, University of Pennsylvania, 1996.

[Lua 07] Scripting Lua. *http://www.lua.org/.* World Wide Web, 2007.

[Macedonia 95] M. Macedonia, M. Zyda, D. Pratt, D. Brutzman & P. Barham. *Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments.* In Proceedings of IEEE Virtual Reality Annual International Symposium, pages 2–10, 1995.

[Macedonia 97] M. Macedonia & M. Zyda. *A Taxonomy for Networked Virtual Environments.* IEEE MultiMedia, vol. 4, no. 1, pages 48–56, 1997.

[Magnusson 02] C. Magnusson, K. Rassmus-Gröhn, C. Sjöström & H. Danielsson. *Navigation and Recognition in Complex Haptic Virtual Environments - Reports from an Extensive Study with Blind Users.* In Proceedings of EuroHaptics 2002, Edinborough, UK, 2002.

[Manninen 04] T. Manninen. *Rich Interaction Model for Game and Virtual Environment Design.* PhD thesis, Oulu University, Feb. 2004.

[Massie 94] T. Massie & J. Salisburg. *The PHANToM Haptic Interface: A Device for Probing Virtual Objects.* In Proceedings of the 1994 ASME International Mechanical Engineering Congress and Exhibition, volume DSC 55-1, pages 295–302, Chicago, IL, USA, November 1994. ASME.

[Maya 07] AutoDesk Maya. *http://usa.autodesk.com/.* World Wide Web, 2007.

[Meehan 01] M. Meehan. *Physiological Reaction as an Objective Measure of Presence.* PhD thesis, University of North Carolina, Chapel Hill, USA, 2001.

[Miller 95] D. Miller & J. Thorpe. *SIMNET: the advent of simulator networking.* Proceedings of the IEEE, vol. 83, no. 8, pages 1114–1123, Aug. 1995.

[Mine 95] M. Mine. *Virtual Environment Interaction Techniques.* Rapport technique TR95-018, University of North Carolina, Chapel Hill, USA, Apr. 1995.

[Mine 97] M. Mine, F. Brooks & C. Sequin. *A survey of design issues in spatial input.* In Proceedings of SIGGRAPH 97, 1997.

[Nemec 02] V. Nemec, Z. Mikovec & P. Slav´k. *Adaptive Navigation of Visually Impaired Users in a Virtual Environment on the World Wide Web.* In User Interfaces for All, pages 68–79, 2002.

[Novint 07] Novint. *http://www.novint.com/.* World Wide Web, 2007.

[Oakley 00] Ian Oakley, Marilyn Rose McGee, Stephen Brewster & Phill Gray. *Putting the Feel in 'Look and Feel'.* In Proceedings of CHI 2000, pages 415–422, The Hague, NL, April 1–6 2000.

[ODE 07] Open Dynamics Engine ODE. *http://www.ode.org/.* World Wide Web, 2007.

[Oore 02] S. Oore, D. Terzopoulos & G. Hinton. *A Desktop Input Device and Interface for Interactive 3D Character Animation.* In Proceedings of Graphics Interface (GI02), 2002.

[Park 99] K. Park & R. Kenyon. *Effects of Network Characteristics on Human Performance in a Collaborative Virtual Environment.* In Proceedings of the IEEE Virtual Reality (VR '99), pages 104–111, Washington, DC, USA, 1999. IEEE Computer Society.

[Payne 07] Max Payne. *http://www.rockstargames.com/maxpayne/.* World Wide Web, 2007.

[Pettifer 99] S. Pettifer. *An Operating Environment for Large Scale Virtual Reality.* PhD thesis, University of Manchester, 1999.

[PhysX 07] Ageia PhysX. *http://www.ageia.com.* World Wide Web, 2007.

[Pouprey 98] I. Pouprey, S. Weghorst, M. Billunghurst & T. Ichikawa. *Egocentric Object Manipulation in Virtual Environments; Empirical Evalutaion of Interaction Techniques.* Computer Graphics Forum, vol. 17, no. 3, pages 30–41, 1998.

[Purbrick 00] J. Purbrick & C. Greenhalgh. *Extending Locales: Awareness Management in MASSIVE-3.* In Proceedings of IEEE Virtual Reality, pages 287–287, Feb. 2000.

[Python 07] Scripting Python. *http://www.python.org/.* World Wide Web, 2007.

[Qt 07] Qt. *http://www.trolltech.com/products/qt/.* World Wide Web, 2007.

[Quax 03] P. Quax, T. Jehaes, P. Jorissen & W. Lamotte. *A multi-user framework supporting video-based avatars.* In Proceedings of the 2nd workshop on Network and system support for games, pages 137–147. ACM Press, 2003.

[Quax 07] P. Quax. *An Architecture for Large-scale Virtual Interactive Communities.* PhD thesis, Hasselt University, June 2007.

[Sallnäs 00] E. Sallnäs, K. Rassmus-Gröhn & C. Sjöström. *Supporting presence in collaborative environments by haptic force feedback.* ACM Transactions on Computer-Human

Interaction, vol. 7, no. 4, pages 461–476, 2000.

[SecondLife 07] SecondLife. *http://secondlife.com/*. World Wide Web, 2007.

[SensAble 07] SensAble. *http://www.sensable.com/*. World Wide Web, 2007.

[Seo 00] H. Seo, C. Joslin, U. Berner, N. Magnenat-Thalmann, M. Jovovic, J. Esmerado, D. Thalmann & I. Palmer. *VPARK - A Windows NT Software Platform for a Virtual Networked Amusement Park*. In Proceedings of the International Conference on Computer Graphics (CGI '00), page 309, Washington, DC, USA, 2000. IEEE Computer Society.

[Shapiro 03] A. Shapiro, F. Pighin & P. Faloutsos. *Hybrid Control for Interactive Character Animation*. In Proceedings of the 11th Pacific Conference on Computer Graphics and Applications (PG '03), pages 455–460, Washington, DC, USA, 2003. IEEE Computer Society.

[Singhal 95] S. Singhal & D. Cheriton. *Exploiting position history for efficient remote rendering in networked virtual reality*. Presence: Teleoperators and Virtual Environments, vol. 4, no. 2, pages 169–193, 1995.

[Singhal 99] S. Singhal & M. Zyda. Networked virtual environments: Design and implementation. Addison-Wesley Pub Co, 1999.

[Slater 97] M. Slater & S. Wilbur. *A Framework for Immersive Virtual Environments (FIVE): Speculations on the Role of Presence in Virtual Environments*. Presence: Teleoperators and Virtual Environments, vol. 6, no. 6, pages 603–616, 1997.

[Smith 00] S. Smith, D. Duke & J. Willans. *Designing World Objects for Usable Virtual Environments*. In Workshop on Design, Specification and Verification of Interactive Systems (DSVIS'00), pages 309–319, 2000.

[Stone 00] R. Stone. *Haptic Feedback: A potted History, From Telepresence to Virtual Reality*. In Proceedings of the Workshop on Haptic Human-Computer Interaction, pages 1–8, Glasgow, UK, Aug. 2000.

[Subramanian 00] S. Subramanian & W. IJsselsteijn. *Survey and Classification of Spatial Object Manipulation Techniques*. In Proceedings of OZCHI 2000, Interfacing

Reality in the New Millennium, pages 330–337, Dec. 2000.

[Swift++ 07] Swift++. *SWIFT++, University of North Carolina at Chapel Hill, http://www.cs.unc.edu/~geom/SWIFT++/*. World Wide Web, 2007.

[Thalmann 99] D. Thalmann. *The Role of Virtual Humans in Virtual Environment Technology and Interfaces.* In Proceedings of Joint EC-NSF Advanced Research Workshop, 1999.

[Thalmann 00] N. Magnenat - Thalmann & C. Joslin. *The Evolution of Virtual Humans in NVE Systems.* ICAT2000, pages 2–9, Oct. 2000.

[Thalmann 01] D. Thalmann. *The role of virtual humans in virtual environment technology and interfaces.* Frontiers of human-centred computing, online communities and virtual environments, pages 27–38, 2001.

[There.com 07] There.com. *http://www.there.com/*. World Wide Web, 2007.

[Unreal 3 07] Unreal 3. *http://unrealtechnology.com/html/technology/ue30.shtml*. World Wide Web, 2007.

[Usoh 00] M. Usoh, E. Catena, S. Arman & M. Slater. *Using Presence Questionnaires in Reality.* Presence, vol. 9, no. 5, pages 497–503, 2000.

[VLNET 07] Virtual Life Network VLNET. *http://ligwww.epfl.ch/~thalmann/vlnet.html*. World Wide Web, 2007.

[Ware 90] C. Ware & S. Osborne. *Exploration and virtual camera control in virtual three dimensional environments.* In SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics, pages 175–183, New York, NY, USA, 1990. ACM.

[Waters 97] R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns & W. Yerazunis. *Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability.* Presence: Teleoperators and Virtual Environments, vol. 6, no. 4, pages 461–480, Aug. 1997.

[Watsen 98] K. Watsen & M. Zyda. *Bamboo - Supporting Dynamic Protocols for Virtual Environments*, Aug. 1998.

[Welman 89a] C. Welman. *Inverse Kinematics and Geometric*

Constraints for Articulated Figure Manipulation. PhD thesis, School of Computer Science, Simon Fraser University, 1989.

[Welman 89b] C. Welman. *Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation.* PhD thesis, School of Computer Science, Simon Fraser University, 1989.

[West 98] A. West & R. Hubbold. *System Challenges for Collaborative Virtual Environments.* In D. Snowdon & E. Churchill, editeurs, Proceedings of Collaborative Virtual Environments (CVE'98). Springer-Verlag, 1998.

[Witmer 96] B. Witmer, J. Bailey, B. Knerr & K. Parsons. *Virtual spaces and real world places: transfer of route knowledge.* International Journal Human-Computer Studies, vol. 45, no. 4, pages 413–428, 1996.

[Witmer 98] B. Witmer & M. Singer. *Measuring Presence in Virtual Environments: A Presence Questionnaire.* Presence: Teleoperators and virtual environments, vol. 7, no. 3, pages 225–240, June 1998.

[Zordan 02] V. Zordan & J. Hodgins. *Motion capture-driven simulations that hit and react.* In Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA '02), pages 89–96, New York, NY, USA, 2002. ACM Press.