

Interfacing Virtual Worlds Based on VRML and MRT

Diplomarbeit

vorgelegt von

Oliver Jucknath

Betreuer: Prof. Dr. D.W. Fellner

Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III
Römerstr. 164, D-53117 Bonn

9. August 1996

Versicherung

Hiermit versichere ich, daß ich diese Arbeit selbständig und ohne fremde Hilfe angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet, sowie Zitate als solche kenntlich gemacht habe.

Köln, den 9. 8. 1996

Danksagung

Herrn Professor Fellner danke ich für die Möglichkeit zur Durchführung dieser interessanten Arbeit und seine hilfreiche Unterstützung. Ebenso danke ich allen Mitarbeitern der Computer-Graphik-Gruppe – insbesondere Martin Fischer und Stephan Schäfer – für die gute Zusammenarbeit.

Ausserdem danke ich meinen Freunden Jan, Martin, Michael, Norbert, Pete und meiner Schwester Susanne für ihre ständige Hilfs- und Gesprächsbereitschaft.

Contents

1	Overview	5
2	MRT	7
2.1	General library	8
2.2	Rendering libraries	8
2.2.1	Form library	9
2.2.2	Scene library	9
2.2.3	Light library	9
2.2.4	Surface library	9
2.2.5	Raytracer library	9
2.3	CGI++	10
2.4	CGI-3D	10
2.5	Penguin	10
3	VRML	11
3.1	Basic structure	12
3.2	3D object description	13
3.3	Hyperlinks	13
3.4	File Inclusion	14
3.5	Future of VRML	14
4	Joining VRML and MRT	16
4.1	VRML Parser	16
4.1.1	Motivation	16
4.1.2	From YACC to PCCTS	16
4.1.3	From Framework to Application Support	17
4.1.4	The Parser Chain	18
4.1.5	Variations of VRML	19
4.1.6	Mapping VRML nodes to MRT objects	19
4.1.7	Handling named objects	22
4.1.8	Alternative parsers	23
4.2	VRML Browser	23
4.2.1	Motivation	23
4.2.2	The internal dispatcher	23
4.2.3	The Interactor	24
4.2.4	Connecting to the internet	25
4.2.5	Connecting to other applications	25
4.3	Enhancing the user interface	25

4.3.1	Motivation	26
4.3.2	The heuristic rules	28
4.3.3	Porting to other 3D packages	36
4.4	Conclusion	37
5	IRC	39
5.1	Users	39
5.2	Channels	39
5.3	Channel Operators	40
5.4	Servers	40
5.5	Operators	40
5.6	Efficiency	41
6	Joining IRC with MRT/VRML	42
6.1	Motivation	42
6.2	Unicasting vs. Multicasting	42
6.2.1	Unicasting	42
6.2.2	Multicasting	43
6.2.3	Using IRC as a distribution protocol	44
6.3	Central vs. Distributed representation	45
6.3.1	Central representation	45
6.3.2	Distributed representation	45
6.3.3	The commercial aspect	46
6.4	Common requirements	47
6.5	The server centered approach	48
6.6	The multicasting based approach	50
6.7	The distributed approach	51
6.8	Conclusion	53
7	Conclusion	55

Chapter 1

Overview

This thesis takes three well known technologies and shows that their synthesis allows to build an interface to virtual realities with new and unique features. These three base technologies are: MRT, VRML and IRC.

Chapter 2: MRT. The MRT [7, 8] is a modular rendering toolkit that provides an object hierarchy which covers most aspects of generating synthetic 3D images, ranging from simple vector arithmetic to sophisticated raytracing and radiosity algorithms. The entire toolkit is available on several platforms including several flavors of Unix as well as Microsoft Windows. This removes the burden of handling the different platform dependent 3D rendering engines, e.g. OpenGL, XGL or 3DR, by providing a platform independent interface.

Chapter 3: VRML. The second technology is VRML [15], a crossover of HTML (RFC 1866, [26, 27]) (the page description language of the WWW) and IV (the OpenInventor [23, 32] file format for 3D Objects). VRML was designed as a language to describe virtual worlds in which objects are associated with hyperlinks. These hyperlinks may point to other VRML worlds or just any information that can be accessed via the WWW. VRML is the first standard for virtual realities that gained acceptance all over the internet. The main reason for its fast success was the fact that it used the hypertext transfer protocol (HTTP) (RFC 1945), thus avoiding the need for new specialized servers, because every HTML server could be used as a VRML server. As a consequence VRML also inherited all the drawbacks of HTML. Therefore, VRML worlds do not have any way of remembering events or states and there is no support for multi-user worlds. But in spite of its many disadvantages VRML was in general enthusiastically welcomed as a first step towards Cyberspace.

Chapter 4: Joining VRML and MRT. The first obvious combination of VRML and MRT was to add a VRML parser to the MRT. In this way, the MRT was able to render any scene which is available in the VRML format. Since there are many converters from different formats to VRML, this significantly extended the set of scenes available to the MRT. The second still obvious step was to write a complete VRML browser, using the MRT to provide a platform independent rendering engine and user interface. With this VRML browser, called MRTSpace, a user can visit a virtual world and access the information associated to the hyperlinks. Because MRTSpace was one of the first stand-alone applications developed based on MRT, it also took over the role as a proof of concept for the object-oriented design of the MRT. Although it was not an intention to improve the browser's behavior in the first place, it turned out that the ability of the MRT to gain a limited understanding of the scene allowed the design of a more advanced user interface. Instead of just handling an unrelated set of polygones, MRTSpace has a set of heuristic rules that handles walls, steps, obstacles, doors or pitfalls in a way a user would expect such objects to behave, e.g. a stair can be climbed, walls are solid etc.

Chapter 5: IRC. The third key technology integrated into this thesis is IRC (RFC1459, [14, 29]), which is the internet's equivalent to ham radio. And just like its physical counterpart, IRC is mainly used to chat with other people and to keep up social contacts over long distances. The main advantage over e-mail is that IRC allows the user to talk in real time to a group of other people. In analogy to ham radio, each of these groups is called a channel. The life time of such a channel is rather short. It starts with a single user, who posts the creation of a channel, mostly with an associated theme. Other IRC user can then join in or leave the channel, but when the last user exits from a channel, the channel will be shut down. This channel allocation process needs a specialized IRC server, in the original swedish implementation a single computer. But as IRC spread all over the world, the IRC backbone was developed, a network of several IRC servers that handle channel allocation. Moreover, the IRC backbone implements a sophisticated message bundling mechanism to minimize network load. This highly efficient message handling is the main reason why IRC could spread all over the world, even into regions with very low bandwidth.

Chapter 6: Joining IRC with MRT/VRML.

IRC was integrated into the MRTSpace to overcome the lack of multi-user support in VRML. The main idea is to use VRML for the static definition of a virtual world and to use IRC to handle the exchange of information about dynamic processes like movements of avatars or user to user messages. Using the IRC backbone has two benefits: There is no need to install new specialized VR servers, and the needed bandwidth is very low. In comparison with networked multi-user virtual reality systems that use multicast packets for broadcasting, using IRC for broadcasting induces additional overhead. The IRC-based approach is therefore less efficient in small, high-bandwidth network environments. However, by obviating the need for multicast transmissions, using IRC for broadcasting allows the realization of a wide area networked virtual reality based on today's Internet in which wide area multicasting is not effectively feasible. The entire multi-user support was accomplished by a small set of basic C++ objects, which are now part of the MRT, thus extending the range of future applications that can be supported by the MRT.

Chapter 2

MRT

The Modular Rendering Toolkit (MRT) [7, 8] it is developed at the University of Bonn by a group led by Professor Fellner, designed to provide support for most aspects of synthetic 3D image generation. The following properties of the MRT were especially helpful for the implementation of the MRTSpace:

Object orientation The MRT was designed from the very beginning to be a large scale real world system based on the object-oriented programming paradigm. Therefore, the entire functionality of the MRT can be accessed by well-encapsulated objects. Because of this separation between the MRT and any application using it, the development of MRTSpace was not influenced in a negative way by the rather fast development of new features for the MRT.

Rich functionality The MRT provides a wide range of objects which cover all the basic needs of any application in the context of 3D rendering. Moreover, it provides high quality rendering algorithms that are aimed to provide photorealistic image generation. An additional package even provides a sophisticated GUI, which simplifies rapid prototyping. The MRTSpace uses only libraries of the MRT package and the standard C libraries.

Platform independence The MRT is available for several flavors of Unix running X11, as well as for Windows 95/NT. MRTSpace was completely developed under Unix. The only necessary changes for the Windows version concerned thread handling, because this functionality was not supported by the MRT.

Symmetrical object handling. The functions and objects of the MRT are as symmetrical as possible. Most functions like animation, raytracing and constructive solid geometry can be performed on any 3D object. As a side effect of this symmetrical structure, any enhancement of a functionality is immediately available to all 3D objects. The MRTSpace uses the methods to display 3D objects in real time. Later on, the need arose to apply raytracing techniques on these 3D objects. Because of the symmetrical nature of the MRT, the same object structure could be used for both: approximation and raytracing.

Extendable The object hierarchy provides many points to add new functionality in a consistent way. For example, the missing polygon object - required by VRML - could be added with less than 100 lines of C++ code, providing all the typical functionality of any other 3D object of the MRT.

The following sections will give a short overview of the libraries of the MRT that have been used to implement the MRTSpace. Some further aspects of the API provided by the MRT will be discussed in chapter 4.

2.1 General library

The general library covers the very basic functionality needed by most applications. It can be divided into three parts. The first part deals with basic tools for computer graphics:

- 2D and 3D vector and matrix arithmetic
- perspective and orthographic projections
- color representation
- surface properties
- ppm file format

The second part contains objects that try to compensate for some of the deficiencies of C++. Some of these objects will become superfluous as soon as ANSI C++ 3.0¹ is widely available, especially the run time type information.

- safe usage of 'new' and 'delete'
- reference pointer
- run time type information
- dynamic lists

The former objects are highly portable. The general library provides the following objects to encapsulate the platform dependent functions. Most of these methods cope with functionality that was not defined by ANSI, or where the ANSI standard left ambiguities.

- command line parsing
- time measurement
- file I/O
- random numbers

Another notable object is `t_Real`, which is the basic object for all floating arithmetic inside the MRT. Because this is a C++ object, a very fine control of the precision and mathematical behavior is possible, for example extended precision or smart handling of rounding errors. For applications which need the highest speed it is possible to replace this object by a typedef to float, which was done for the MRTSpace.

2.2 Rendering libraries

These libraries form the inner kernel of the MRT. They contain all the 3D objects, surface definitions, shading models, and raytracing or approximation methods.

¹<http://www.cygnus.com/misc/wp/draft/>

2.2.1 Form library

This library contains all the 3D objects. Besides the common basic shapes, this library contains support for:

- CSG, which supports intersection and union of any 3D object
- Meta balls
- solid of revolution
- Beziers and Splines

Each of these 3D objects is associated with a C++ object, which has methods that enable the image objects to display themselves in approximated or raytraced form. MRTSpace uses most of these objects in their existing form, only a 3D object representing a polygon had to be added.

2.2.2 Scene library

This library supports the different raytracing techniques: hierarchical, octtree and voxel. These rendering techniques are associated the scene concept of the MRT. The MRTSpace uses one of the raytracing methods – the intersection of a ray with a 3D object – to implement a proximity sensor that guides the user movement.

2.2.3 Light library

The light library contains the different light objects: spotlight, pointlight and directional light. As the shadowing algorithms are bound to the light objects, this library contains also the methods to support sharp or smooth shadows. Because the light models of the MRT and VRML differ slightly, some minor extensions had to be performed.

2.2.4 Surface library

This library contains the basic surface definition and several derived classes: Bump mapping, solid texturing and bitmap textures. Most of these classes are available both for approximated and raytraced rendering techniques.

2.2.5 Raytracer library

This library contains the image object, which provides methods to support the generation of either raytraced or approximated images. Moreover, it contains all the basic control and I/O objects:

- General control object
- Parser chain
- Parser for the Modular Scene Description language (MSD)

Furthermore, it contains all the basic objects required to support the B-Reps, which replaced the former triangle strip list. As part of this work, some of the control and parsing methods have been extended (see Chapter 4).

2.3 CGI++

CGI++ [9] is a stand-alone API that provides device independent 2D rendering and simple support for input devices. The current implementation supports several Unix variations and Windows 95/NT. CGI++ has several internal layers of abstraction, which allows to adapt to the different levels of available hardware support for each platform. To provide platform independent fonts, CGI++ supports the True Type standard.

MRTSpace uses CGI++ to overlay 3D graphics with text. An example for this is the in screen display of text spoken in a virtual chat room. Moreover, the in-screen control elements are drawn with the CGI++ graphic primitives.

2.4 CGI-3D

CGI-3D [10] is the three dimensional counterpart of CGI++. It provides a platform independent way to display 3D graphics and animations. CGI-3D supports the different available rendering engines on each platform. The current version supports OpenGL on SGI's, XGL on SUN's and 3DR on Windows 95/NT. For other platforms, a software rendering engine exists. To support different rendering engines, CGI-3D has to translate objects from the internal MRT format to their counterparts in each of the rendering engines. This separation of the internal format from the platform dependent rendering engines reduced the cross platform development of the MRTSpace in a significant way. CGI-3D relies on CGI++ to perform several platform dependent tasks, e.g. opening or closing windows, and is therefore available on all platforms that are supported by CGI++.

2.5 Penguin

Penguin [11] is a compact and portable graphical user interface. Although Penguin is not a part of the MRT package, it provides optimal support for any application developed on the basis of the MRT, because Penguin is based on CGI++ and therefore has a simple method to integrate CGI-3D. Moreover, Penguin is one of the truly portable systems that spans the gap between Unix and Windows. It provides a consistent look, independent of the platform. MRTSpace uses Penguin to implement its menus, dialog boxes and the main screen buttons. One of the strong points was the handling of user generated events. Penguin accepts user generated events even if they are generated from a child process. As MRTSpace creates multiple processes, this feature was highly useful.

Chapter 3

VRML

The Virtual Reality Modelling Language (VRML) [2, 1] is intended to evolve into the Internet standard for multi-user virtual realities. This evolution is planned to take the following steps:

1. VRML 1.0 released in 1995 [3] was designed to provide a language to describe the very basics of a single user virtual reality.
2. VRML 2.0 released in 1996 [4] added support for animations and external scripting languages.
3. VRML 3.0 is expected to add multi-user support.

As this thesis is based on VRML 1.0, the following introduction will focus on this version. Moreover, the additions of VRML 2.0 are independent of the extensions proposed in this work.

VRML was designed to meet three requirements:

Platform independence VRML was intended to develop into the main standard for all multi-user virtual worlds in the Internet and was therefore required to be portable to any available platform. The central point was to avoid any copyrights or royalties. Especially the heavy influence taken by SGI provoked several critical comments in the first design phase. SGI reacted to these critics by renouncing of any copyrights on VRML, although VRML is a subset of SGI's proprietary file format called IV.

Extensibility As VRML is a very fast evolving standard, the need to incorporate extensions to the language, without the need to redefine the entire language, was essential. The basic structure for a VRML file provides this extensibility, because it uses a keyword oriented system of nesting data blocks. As the length of the block is easy to determine by finding the matching brackets that enclosed every block, it is easy for a parser to skip data which is associated to an unknown keyword.

VRML also introduced a concept to describe the exact syntax of these extensions, but this approach proved to be superfluous, as long as no semantic information could be attached to the syntax of a new data block.

Efficiency VRML was designed to work even over lines with a low bandwidth. The main idea of VRML is to send a description of the 3D objects and render them on the local client. The alternative is to render on a central server and use the clients only to display the images. Although this approach has several advantages it will usually require a permanent client-server connection with a high bandwidth. The VRML file is transferred only once and all

following actions can be done locally. Even if the bandwidth is very low, only the time to load the VRML file is extended, the speed of the following interaction depends only on the computation power of the client.

The first practical tests with VRML exhibited a weakness of VRML: because it is based on IV, it is a human readable clear text format, and all numbers are send in a plain readable ASCII format, which extended the filesize by a factor of at least two compared to a binary representation.

Another drawback evolved over the time: As soon as converters from other file formats to VRML were available, a considerable part of the VRML designers switched from small hand coded VRML files to large automatically converted files, consisting of endless triangle lists. The typical file size exploded to something between 100 KBytes and several MBytes. This increased the loading time over typcial Internet connection into painful dimensions. The underlying problem is that the typical converter takes the high level description of the input format and transforms it to a series of triangles, because VRML has no counterpart for the high level description of the input file. Although the extension of the basic set of shapes available in VRML - especially splines and nurbs are missing - would be helpful, the basic problem of interchanging 3D objects on a high level of abstraction is still to be solved.

3.1 Basic structure

VRML is a subset of IV (the OpenInventor file format) with two extensions to support hyper links and file inclusion. Although IV was designed as a 3D file format, its basic structure is suitable to store any data that has a tree structure. The basic element is a node. Each node starts with a name, which defines the type and meaning of this node. This name is followed by a pair of matching curl brackets. Inside these brackets there can be two kinds of data: fields and subnodes. Fields consist of a name, which defines the type and meaning of the field, followed by the value of this field. The order of the fields is arbitrary, the meaning of any value is determined by the preceding name. As every field has an associated default value, all fields are optional. There are no special field separators. Because a name will never be a value, every name marks the beginning of a field. If a list of elements is required as a value square brackets are used to mark its start and end.

```
Sphere {} // A sphere object, with no fields,
           // default radius = 1.0
```

```
Sphere { radius 2.0 } // the field radius defines both type and meaning
```

```
Box { width 1 height 2 }
Box { height 2 width 1 } // two identical objects
```

```
IndexedFaceSet { index [1, 2, 3, -1, 2, 3, 4, -1] } // list as a value
```

Some nodes accept subnodes inside the curl brackets. Subnodes have the same form as nodes and subnodes may be arbitrarily nested. If more than one subnode is present, the order of these subnodes is sometimes of importance.

```
LevelOfDetail
{
  distance [ 20 ] // if the user is less than 20 units away take the
  Sphere {} // first subnode, otherwise the second subnode.
  Box {}
}
```

3.2 3D object description

VRML has a set of basic shapes, which are a very compact way to describe 3D objects: Sphere, Cone, Cylinder, Text and Box. Unfortunately these forms are used only in hand-coded scenes, most converters produce just triangle lists. These triangle lists are translated to the VRML object called IndexedFaceSet. An IndexedFaceSet consists of an arbitrary number of convex polygons. Each vertex of a polygon can have an associated normal to allow for the approximation of round objects. The term 'indexed' stems from the IV concept of indexed objects. An indexed object can have a material description associated with each face or vertex. There is no restriction for the vertices and faces of an IndexedFaceSet, especially the faces do not have to be continuous. Some modellers even use a single IndexedFaceSet to store the entire scene.

The basic way to describe the surface characteristics of an object is the material node, which supports all the basic characteristics of optical behavior, with the notable exception of reflectiveness. Each object may have a texture associated, which can be inline coded as a field value, but is most times only referenced by a filename. It is up to the VRML browser to get this file, which contains the actual texture. All texture maps will change only the diffuse reflection parameter, all other parameters are taken from the material node.

A remarkable feature of VRML is the LevelOfDetail node, which allows to assign several representations to a single object. Depending on the distance to the viewer, one of these representations is chosen. Typically a very coarse approximation of the objects shape is chosen to be displayed if the user is far away, and a very fine detailed version of the object is shown only if the user is near enough to appreciate these details. This mechanism allows to create large and heavily detailed scenes, which can be rendered even on a low level client. Although very helpful, this feature is used by only a very few VRML designers who still hand-code their scenes.

Furthermore, VRML has three different kinds of light: PointLight, DirectionalLight and SpotLight, two different kinds of camera: PerspectiveCamera and OrthographicCamera, and several types of nodes needed to group objects.

The selection which IV objects should be included into VRML had some questionable aspects: Some VRML nodes are hard to port to most other 3D packets (e.g. MaterialSeparator) or are of obscure usability (e.g. OrthographicCamera). On the other hand support for splines or nurbs has been omitted. Although this simplified the porting to rendering engines that lacked such functionality, it forces a VRML converter to generate a large number of approximating triangles, where a short nurb description would be sufficient.

3.3 Hyperlinks

The WWWAnchor node can associate an URL to any 3D object. This URL is used when the user activates the associated 3D object. The VRML standard does not specify what an 'activation' is, but most VRML browser interpret a click of the user onto a 3D object as an activation. There are several types of VRML browser and each of them handles an activation in a different way:

1. A plug-in forwards the URL to the master application. This has an unpleasant effect if that URL points to another VRML page: The window opened by the plug-in will be closed as the plug-in terminates and reopen as the master application starts the plug-in again to show the new VRML page.

2. A semi-independent program will call a program that can handle MIME extensions only if the URL to be followed points to anything other than a VRML page. There are several programs that cooperate with netscape or mosaic in this way.
3. A stand-alone application will handle any MIME itself or start the appropriate helper applications.

There are two most common uses of the WWWAnchor node: either to interconnect different VRML pages to form a larger virtual reality or as a connection to an HTML page, which is a much better means to transport written information than VRML itself.

3.4 File Inclusion

The WWWInline node allows for the inclusion of other VRML files by giving the appropriate URL. Although simple in design, this mechanism is very useful.

The first method to use this node is to split the main scene into essential and optional objects. The essential objects are hard-coded into the scene, while the optional objects are put in an include file. A smart browser can now allow the user to interact with the essential objects, while it is still reading the optional objects. The optional objects will be shown only as a bounding box, which will be replaced by the actual object as soon as it is available.

In combination with the LevelOfDetail node the include mechanism allows to implement a 'load on demand'. The LevelOfDetail node has a coarse description of the 3D object as its first subnode and a WWWInline as its second subnode. The first node is chosen when the user is far away and the detailed version stored in the file the WWWInline points to is only loaded if the user comes near to this object. A smart browser may even start reading the include files inside the LevelOfDetail node as soon as this node is parsed.

The most important aspect of WWWInline is the ability to build libraries. The Internet community is expected to develop several sets of reusable objects. These basic objects could be referenced by a WWWInline node. A smart browser (or even a smart external network cache) will have a way to avoid the reloading of these objects, and will provide some kind of cache. If there is a sufficiently sophisticated set of such libraries well known by most clients, VRML files would consist mostly of references to these library objects, thus combining highly detailed objects with a low net load. But today only the very first beginnings of these libraries have been seen on the Internet.

3.5 Future of VRML

The official successor to VRML 1.0 is the MovingWorlds proposal of SGI. Although there are some minor syntax changes, it is mostly upward-compatible to VRML 1.0. The major addition is the introduction of animations and the support for external scripting languages. Simple animation can be created by linking a generator node to any field of a 3D object. This field will change whenever the associated generator node changes its output value. There are several generator types, some will simply switch between several different values at a fixed frequency, some will interpolate between such values, others can react in response to user actions.

Although these generators allow the construction of complex animations, there will sometimes arise the need for the expressiveness of a full programming language. VRML 2.0 has an interface to external scripting languages, which allows to treat any script as a generator node. The VRML 2.0 specifications do not prefer any scripting language, although the Internet community seems to

choose Java [20]. Using external scripting languages has several advantages: It saves the otherwise required development time needed to define and implement a programming language, it defers the actual decision which language to use until some experience in this field is acquired, and it allows to change the used programming language without the need to redefine the language itself.

Furthermore, it is reasonable to argue that there is a vast number of existing programming languages to choose from and it is not necessary to further multiply the number of programming languages.

The drawback of this approach is that it requires the designer of a VRML scene and the user who examines the VRML file to agree on a set of programming languages. If the VRML file contains a script for a language that is not available on the client machine, it can not be displayed. As Java is expected to be widely available, there may be a strong tendency to use Java as the only scripting language. But if Java is the only language to be used with VRML 2.0, it would have been wiser to provide a direct integration of Java, because such an interface could have been much smaller than the elaborate general purpose interface of VRML 2.0.

One of the rejected proposals for the VRML 2.0 standard was the ActiveVRML of Microsoft [33]. Its main advantage is a seamless integration of a functional programming language, but it was in no way upward-compatible to VRML 1.0. Most of the groups discussing the development of VRML were skeptical towards ActiveVRML, because it contradicted the basic philosophy to reuse existing technologies wherever possible. The MovingWorld proposal of SGI was based on IV, which has been in use for several years, and the integration of existing programming languages. ActiveVRML proposed a new programming language together with a new 3D object format. Moreover Microsoft has – in contrast to SGI – no long term experience with 3D technologies, so the final vote for VRML 2.0 was also influenced by the image of the companies proposing the standards.

A completely different approach that may end VRML in its existing form is the proposal to avoid the rather clumsy interface between VRML 2.0 and Java by providing a class library which provides the functionality to describe 3D objects. This approach is the reversal of VRML 2.0: Not Java is to be integrated into a 3D language, but a 3D object library into Java. A good starting point is the OpenInventor library of SGI, which provides the same functionality as IV. Such a 3D library for Java would provide a seamless integration of programming language and 3D object description.

Chapter 4

Joining VRML and MRT

4.1 VRML Parser

4.1.1 Motivation

The implementation of a VRML parser into the MRT was the first step towards the development of a complete VRML browser. The ability to interpret VRML will be used later on in two different ways: to parse VRML files that contain the static definition of a virtual reality, and to parse the content of IRC messages which describe the dynamic processes in the virtual reality. Furthermore, the implementation of this parser is a required step to build a platform for experiments which involve altering the language definition of VRML itself. VRML is considered to be the first step towards establishing a standard for the description of virtual realities. In further steps, more advanced programming features will most likely be added. To be prepared for such an evolution, it is important to choose parser generating tools that allow dealing with object descriptions up to the level of a complete programming language.

As a welcomed side effect the set of scenes available to the MRT was extended. The original MRT had only one parser for a language called MSD (Minimal Scene Description language). There were no converters from other 3D formats to MSD, so that all test scenes for the MRT had to be hand coded ASCII text. For technical testing this was fully sufficient, but as the MRT developed further towards the goal of photo realistic rendering the need to be able to process scenes generated by a 3D editor became more pressing. Although yet there are only a few 3D editors that directly support VRML as a file format, there is a host of converters that translate several file formats to VRML. In this way it is also possible to compare the MRT with other rendering tools by rendering general reference scenes.

4.1.2 From YACC to PCCTS

The original MRT used FLEX as a scanner and YACC [6, 17] as a parser for the MSD language. This approach had several advantages: FLEX generates very fast finite state machines that use down to 12 machine instructions per byte scanned. The source code for FLEX is cryptic, but once understood, it is very easy to maintain. YACC takes a grammar file and converts it to a push down automaton, which is very efficient both in time and space. The source code resembles the Backus Naur Form and is easy to read. But code quality was not the primary reason to choose the FLEX/YACC combination, much more important was the low overhead of adding a new object to the parser. Using the FLEX/YACC combination made it easy for programmers (who, in practice,

often have been students) to integrate new objects into the parsing mechanism, allowing them to focus on the implementation of the object itself.

The drawback of using FLEX/YACC is the introduction of a design weakness into the MRT: Both are not object-oriented and it is very hard to fit them into a clean object-oriented design. The main problem is that the parser generated by YACC uses global variables to propagate the result of the parser run to the main program. These global variables lead to a series of unexpected dependencies. Most of these dependencies do no harm as long as there is only one single scene interpreted by a single parser, but for the introduction of a second parser for VRML and the handling of multiple scenes needed by the VRML browser, this part had to be redesigned.

With the requirements of being object-oriented and non-commercial, the only choice for a parser generator is PCCTS¹. It has several advantages over the classic FLEX/YACC combination:

- Parser and scanner provide an object-oriented interface to the main program.
- One single source file describes both the scanner and the parser.
- The syntax description language is very close to the extended Backus Naur form.
- The parser has an arbitrary number of look-ahead tokens.
- It is easy to extend a top-down parser, because every rule is translated into a virtual method.
- The rules have a powerful argument-passing mechanism.

On the other hand, PCCTS has several drawbacks:

- The scanner it generates is much slower than the one generated by FLEX. PCCTS therefore offers an interface to integrate FLEX, but this necessitates two source files.
- Some conflicting rules are only detected at runtime and cause the program to terminate.
- Some features are not well implemented and lead to unexpected error messages.
- There is no way to intercept the error output of the parser. It invariably goes to the standard error stream. Moreover, it is impossible to add information to an error message, e.g. the name of the parsed file.

The main reason for both its advantages and disadvantages is that PCCTS is a very young product and is evolving quite fast, at least when compared to FLEX/YACC, which accompanied the Unix operating system since its very first days and are no longer modified, as they are considered to be bug-free. But in the long run, PCCTS will most likely outperform FLEX/YACC in most aspects, so PCCTS was chosen to be the new parser tool for the entire MRT and the existing MSD parser was redesigned to use PCCTS.

4.1.3 From Framework to Application Support

The MRT was originally designed as a framework for rendering algorithms. It has a sophisticated object hierarchy, which provides several points for inserting new objects in order to increase the overall functionality of the MRT. This hierarchy was embedded in a fixed main program, which took care of all user I/O and interconnected the parser with the rendering objects. The initialization

¹<http://www.parr-research.com/~parrt/prc/>

of all this was handled by a single setup function that received the command line as its only parameter. Furthermore, the main program was built to handle only a single scene, which allows a very efficient handling of texture memory.

Programming of applications that involve repeated cycles of parsing and rendering or even handling multiple scenes simultaneously was not adequately supported by this simplistic interface. To better support writing such applications, an advanced, object-oriented interface was designed that allows the individual initialization of multiple parsers/renderers and a clean distinction between the information extracted from a scene description file and the parameters controlling the rendering. This interface consists of three types of objects:

1. All information that controls the rendering process is encapsulated into a single 'control object'. As this object also describes which rendering technique is to be used, each such object is able to describe a complete rendering engine. Therefore, several initialization calls were provided that match one of the typical rendering engines, e.g. raytracing, radiosity or approximation. Further information how to set up the system can be gathered by parsing the command line or querying the underlying operating system for default values. To resolve potential conflicts between several sources of information for a single variable, this object includes an elaborate priority system.
2. All information that describes what is to be rendered is encapsulated into a 'full scene' object. This includes all the 3D Objects of the scene, lights, camera, textures and additional information.
3. For each input file format, there is a corresponding parser object. The information gathered while parsing a file is mainly returned in form of a full scene object. Additionally, some languages include mechanisms to alter the way a scene is to be rendered, so these changes are applied to a control object for further usage.

All objects have several virtual methods that allow easy adaptation to special needs. For example, there is a sophisticated registration system to add user specific parameters to the command line parsing or to handle environment variables.

Altogether, these modifications constitute a significant step in developing the MRT's programming interface from a framework of rendering algorithms into a true API.

4.1.4 The Parser Chain

Although the object-oriented design of PCCTS allows the coexistence of several parsers at runtime, there is still a central point required that decides which of the given parsers will be used. Instead of creating a central function for guessing the format of the input, the parser objects were designed to include a method which returns an estimated probability with which it will be able to parse a given input stream. To estimate this probability, the parser could scan for keywords (e.g. a VRML scene always starts with '#VRML') or look at the optional file extension (e.g. all MSD file names should end with '.msd'). The parser chain itself has two calls related to parser choice: one to register a parser and one that first asks each registered parser for the probability with which it will be able to parse a given input stream and then call the 'parse' method of the parser that returned the highest probability.

Generally, scene description files can contain instructions to include data from other files. The parser chain supports inclusion of files with different data formats by providing a backward call to the parser chain itself. Parsers can use this callback to parse files to be included, thus allowing the mixture of different languages to describe a single scene.

Although this is a very handy feature, its use is limited by the fact that no parameter passing across language borders is possible, because even the very concept of what a parameter or name is varies strongly from language to language.

4.1.5 Variations of VRML

The VRML standard has been defined by the VRML 1.0 reference manual. Unfortunately, this paper left several technical aspects of the implementation open. Even after several clarification papers, some essential questions are still not answered. Therefore, most VRML browsers and scene authoring tools used WebSpace, the first VRML browser, as a reference implementation. This resulted in the creation of VRML tools that are not entirely compatible with the VRML 1.0 standard, because WebSpace itself does not conform to several guidelines of the VRML 1.0 standard. Furthermore, the parser of WebSpace is only a patched version of the OpenInventor IV parser with minor additions, thus WebSpace still understands the complete IV language, which is a large superset of VRML. A significant number of authors started using these IV features, e.g. animation features, in their scenes, and – possibly not being aware that the features they used are not part of the VRML standard – posted these scenes as VRML files.

An additional problem is the interpretation of the DEFAULT keyword used in several nodes. Although the documentation explicitly warns that the usage of the DEFAULT keyword leads to a behavior that varies with the different implementations, several scenes were designed that render only in a proper way if the interpretation of the default behavior matches exactly that of WebSpace.

Altogether, this resulted in several different conceptions of the VRML standard. The implementation described here tries to adhere as closely as possible to the original VRML definition and relies on the reference implementation only where the paper does not provide the required information. Particularly, the interpretation of the DEFAULT keyword matches the behavior of WebSpace.

4.1.6 Mapping VRML nodes to MRT objects

The next level of problems surfaced when trying to find pairs of corresponding objects in VRML and MRT, which are based on different philosophies. VRML is derived from OpenInventor, which is built on top of OpenGL [19, 22] as a 3D engine. The basic philosophy of OpenGL is to give the user a maximum of functionality, as long as this can be done in real time and in a portable way. This leads to a host of calls and techniques on several levels of abstraction. The lowest level supports the pixel on screen as an atomic object.

MRT, on the other hand, originates from the world of raytracers, where the quest for photo realism leads to the implementation of sophisticated mathematical models. In most cases, both approaches lead to convergent types of objects and programming interfaces, but there are divergent features that are hard to map from one system to the other. For example, OpenGL has no support for reflective surfaces, as this currently cannot be done in real time. On the other hand, MRT in its orientation towards solid objects has no 3D pixel or line object. But in most cases, it was possible to create a new object that fit well into the MRT hierarchy to implement the missing features.

Depending on the type of node, there are four different ways to map them from VRML to MRT: Some objects have direct counterparts, some objects modify the internal stacks of the VRML browser, some are ignored and the hyperlinks are handled in a special way.

Directly mapped objects

- `Ascii` is a 3D object that describes a string of letters with several attributes, e.g. bold, italic,

or different font styles. This object is mapped to the `t_3Dstrg`, although `t_3Dstrg` does not support any attributes nor all ascii characters. A new `t_3dstrg` based on True Type fonts is expected to replace this version in the near future.

- `Cone` is a 3D object. Both the coat or the bottom may be omitted. Therefore it was mapped to a combination of `t_CylinderCoat` (top radius=0) and `t_Disk`.
- `Cube` is a 3D object. Because any side may be omitted, it is mapped to a combination of six `t_Quadrangles`, which also simplifies the correct texture mapping. Because this is a very frequent object, there is a short cut: If no textures are used and all six sides are activated `Cube` is mapped to `t_Box`, which is considerably faster than the compound of 6 `t_Quadrangles`.
- `Cylinder` is a 3D object. Either bottom, top or coat may be omitted. Therefore, it was mapped to a combination of `t_CylinderCoat` and `t_Disk`.
- `DirectionalLight` was mapped to a `t_Light`, which is positioned very far away from the scene, to emulate a light source that emits parallel light rays.
- `Group` was mapped to `t_Scene`. This way, the hierarchy of a VRML file is mapped to an object hierarchy inside MRT.
- `IndexedFaceSet` describes a set of 3D polygons and was implemented as a compound of `t_NormalTriangles`. The splitting of polygons into triangles introduces some overhead, but as this is exactly what OpenGL does with a `IndexedFaceSet`, this approach allows an exact implementation of the various binding algorithms.
- `LevelOfDetail` was mapped to a `t_Scene` selecting always the highest level of detail. As soon as the level of detail concept of the MRT is ready, this object will be mapped accordingly.
- `Material` is mapped to the `t_SurfaceSpecTrans`. Furthermore, there is an internal cache implemented to make use of MRT's ability to share surfaces between different 3D objects.
- `OrthographicCamera` was mapped to a `t_Camera`, with projection mode set to `PM_PARALLEL`, but the result never looked the way the VRML documentation specifies. These problems seem to arise in the underlying OpenGL layer. Because `WebSpace` – a kind of reference implementation – encountered the same problems, VRML files normally do not contain an orthographic camera.
- `PerspectiveCamera` was mapped to a `t_Camera`, with the projection mode set to `PM_CENTRAL`.
- `PointLight` was mapped to the equivalent `t_Light`.
- `Separator` was mapped to `t_Scene`. This way, the hierarchy of a VRML file is mapped to an object hierarchy inside MRT.
- `Sphere` was mapped to the equivalent `t_Ellipsoid`, because the canonical `t_Sphere` does not respond to transformation matrices in the expected way (it always remains a perfect sphere).

- `Spotlight` was mapped to the `t_SpotLight`. As a consequence, the drop of rate field is not supported, because `t_SpotLight` has no such feature.
- `Switch` is implemented according to the documentation as a selection function. Some VRML browsers use this node in conjunction with the `Info` node to build a list of camera positions.
- `TransformSeparator` is treated in the same way as a `Separator`. The original concept of this node is to separate the transformation scope from the scope of visibility. As this is a very special feature of OpenGL, this feature is practically no more in use and was removed in the 2.0 Specification.

Stack-modifying objects

There are several internal stacks inside the VRML parser which together describe the current state of the parser. This state determines the material and transformation to use when instancing a 3D object and special informations for corresponding objects. For example, a `Normal` node contains the normals to be used by a `IndexedFaceSet`. The following nodes were implemented this way:

- `Coordinate3`
- `MaterialBinding`
- `MatrixTransform`
- `Normal`
- `NormalBinding`
- `Rotation`
- `Scale`
- `Texture2`
- `Texture2transform`
- `Transform`
- `Translation`

Unmapped objects

- `MaterialBinding` was implemented for all modes but `PerVertexBinding`, because most MRT objects do not support this mode.
- `Font` was not implemented due to the lack of expressiveness in `t_3dstrg`.
- `IndexedLineSet` and `PointSet` are not implemented because they have no counterparts in the MRT – the MRT has no concept of pixel oriented objects.
- `Info` nodes are ignored, they are used only to store comments.
- `ShapeHints` are ignored, because the MRT has no way to take advantage of this kind of information.

Hyperlink objects

`WWWAnchor` and `WWWInline` are both handled in a special way. They represent references to other objects via an URL, not actual objects. This URL is stored in a list together with the object that the reference is bound to. This list is then returned as part of the full scene, and has to be processed by the actual browser. An alternate approach is to handle the include statements inside the parser object, but this would introduce several dependencies, linking the highly portable parser to the machine dependent method of building an internet connection.

4.1.7 Handling named objects

The VRML standard has a way to create multiple instances of the same object using the DEF/USE mechanism. The following example produces two adjacent balls.

```
DEF ball Sphere {}
Translation { translation 2 0 0 }
USE ball
```

Unlike in many other languages, the definition is already the first instantiation. There are three different ways to handle this type of referencing.

Resolving instantiation by a preprocessor:

The most obvious solution is to use a preprocessor, for example `m4`, to replace all references to an object with the definition of the object, and then start the parsing process. Although easy to implement, this method may lead to excessive temporary file sizes, prolonged parsing time and an unnecessarily large object tree.

Resolving instantiation by a reference object:

For every definition a template object is created, and for every instantiation, a reference object – which points to the template object – is created. Reference objects are still in the prototype phase. They allow multiple references to a single object without multiplying the corresponding resources. In the current implementation, objects referencing the same template object may vary only in the applied transformation matrix. Future versions will also allow to override the surface or other parameters. But this approach limits the DEF/USE mechanism to the object resolution level of the MRT, which is more coarse than that of VRML. For example, the following definition could not be bound to a MRT object, because there is no concept of normals not associated to any object:

```
DEF MyNormals Normals { point [ 1 0 0, 0 0 1 ] }
```

Combining both techniques:

The approach used in this work is a combination of these two methods. Where a DEF node defines a complete MRT object, a template object is generated and all references are resolved by generating references pointing to this object. Otherwise, the definition is stored in a tokenized form, and a complete new instance is generated at every reference. The major drawback is that this decision can be made only after parsing the node. To avoid reparsing, the parsed information needs to be stored in a C++ object. This necessitates one class per VRML node, thus inflating the source code. As a side effect, this tokenized form turned out to be a valuable source of debugging information for debugging the default behavior, as it allows separating the scanning, parsing and default substitution process from the mapping of VRML to MRT objects.

4.1.8 Alternative parsers

In 1995, SGI published the source of a VRML parser [24] (known as QvLib), which was based on a finite state machine. Although this approach served well for the 1.0 version of VRML, it is not suitable as a platform for experiments with the language itself. In particular, the extension of VRML towards the inclusion of a complete programming language would be impossible on the basis of a finite state machine.

Another implementation, which overcame many of the deficiencies of the SGI parser, was a parser based on the FLEX/YACC combination by Emmanuel Frécon and Olof Hagsand in 1995 [12]. Although this parser could have saved development time at the beginning, the advantages of the object-oriented approach by PCCTS were considered to outweigh this advantage.

4.2 VRML Browser

4.2.1 Motivation

Being able to parse a VRML file into a set of C++ objects for rendering captures only one half of the VRML concept. It does not address the issue of interactivity, which is the main core of any virtual world. Therefore, the next logical step was to build a complete browser which lets the user interact with the world described by a VRML file. The major way of interaction is moving around and examining the different objects of a world. The existing VRML browsers show a wide variety of controller devices. Some use simple arrows, some try to simulate an analogous device and some use even 3D objects to emulate a joystick which can be controlled by a mouse on screen. This diversity shows that this is still an evolving field, where none of the possible methods has gained a clear advantage. To be able to take part in this evolution the functionality of the MRT was extended, towards a framework that allows experiments with interactors. Based on this a complete VRML browser was implemented.

4.2.2 The internal dispatcher

The browser itself can be divided into three parts: the parser, the dispatcher and the interactor. The interactor receives an URL of a new world to be shown from the user and returns it to the dispatcher, which starts a child process to load the file associated with this URL. If this URL points to anything that is not a VRML page, an external viewer will be called. If the URL points to another VRML world, this file is parsed and the resulting scene is handed over to the interactor. Moreover, the dispatcher starts a child process for each include statement. These children try to download the corresponding VRML files. When they received the complete file, they interrupt the interactor and call the dispatcher to re-parse the scene. The dispatcher will then again call the interactor with the updated scene. This approach lets the user interact with the scene as soon as possible, even if some parts are still missing.

There are two difficulties to handle: The first is the platform independent creation of threads and the interrupting of the interactor whenever a child returns. The two platforms supported by the MRT are Unix [31] and Windows [35], which differ greatly in the concept of multitasking. To bridge this gap, a small process library was implemented, which allows a system independent way of creating processes on different platforms. The drawback of this library is that it cannot hide the differences completely, especially the handling of global variables. Therefore, it is possible to write programs with this library that do run perfectly on one platform, but not at all on another. Although this is not satisfactory, the only alternative is to write two different main programs for Unix and Windows.

The second difficulty is the interruption of the interactor when one of the children retrieved an include file. Most of the times the interactor will be in a blocking wait call for new user events. The easy solution is to avoid this blocking call at all and use busy polling, which is easy to interrupt. But this leads to a high CPU load even if the program is idle. Therefore, a more sophisticated approach was used: The child generates a user event that is routed to the underlying window handling system, which propagates this user event to the interactor. Upon receiving the user event the blocking wait call is terminated and the interactor can handle the control back to the dispatcher.

4.2.3 The Interactor

The interactor is an object that, given a full scene, lets the user interact with this scene. The base class of all interactor objects is `t_BaseInteractor`. This class defines only very basic movements, but has a host of virtual methods that can be overridden to implement any application specific behavior. The available methods fall into several groups:

Movement group These methods are called whenever the user activates any kind of movement control. There are four degrees of freedom: up-down, left-right, forward-backward, rotate left-right. These methods usually modify the current camera position by calling a method of the action group.

Preference group These methods set control variables that are typically queried by the movement group. There are parameters for speed, maxspeed and acceleration for each degree of freedom.

Controller group These methods handle the position and layout of the pseudo controller device. This device is a 3D object that appears inside the 3D view and shows an array of arrows. If the user clicks on one of these arrows, the corresponding movement method will be called. In most cases, either the 2-degree or 3-degree devices will be used. Furthermore, there are methods to bind keys to movement actions. This feature may be used to support arrow keys to navigate a scene.

Button group These methods both build and handle the button bar at the bottom of the screen. By inheritance, a user may add buttons and define their behavior.

Menu group In equivalence to the button group, this group defines the menu associated with the interactor.

Mouse group These methods handle the mouse moves, clicks or drags inside the 3D view. These methods can be used to implement some kind of interaction with the 3D objects themselves. A typical application is the highlighting and selection of hyperlinks in a VRML scene.

Action group This group contains utility methods called by the movement group to change the current camera position. The default methods are straightforward, but a more sophisticated version will use them as a key entry point to handle collision detection and other constraints.

Internal group These protected methods are used to provide controlled access to several variables: camera, scene and preferences.

Fall back group This group exists only as a fall back, in case that the former methods did not cover all needed aspects. It provides entry point for constructor, destructor and several other critical entry points.

The first step towards a VRML browser was to inherit from `t_BaseInteractor` and add the functionality to access hyperlinks. This object is called `t_LinkInteractor` and features two modifications:

If the mouse is inside the 3D view and positioned on top of a 3D object that is associated with a hyperlink, this object is highlighted and the URL of this link is displayed at the top of the 3D view. There is a common agreement to use gold as a highlight color, most scene designers will therefore avoid this color. The current implementation uses gold as a highlight color, but if the original diffuse reflection color is too close to yellow a light blue is used as the highlighting color.

If the user clicks onto a highlighted object, the selected object will flash twice, and the interactor returns the URL of the selected object to the calling program.

The `t_SolidInteractor` is derived from `t_LinkInteractor`. It adds routines for handling collision detection and other enhancements, which are described in detail in section 4.3.1. There are two classes derived from this class: `t_WalkInteractor` and `t_FlyInteractor`, which implement the walk and fly modes respectively. These three objects are embedded in the `t_VRMLInteractor`, which supports an easy way to switch between the default, walk or fly mode.

4.2.4 Connecting to the internet

The procedure of retrieving VRML pages over the Internet is well standardized by the HTTP (RFC 1945). Furthermore, both Unix and Windows support the socket library, so there are no difficult portability problems. As this point is of no major interest, the current implementation is rather simple (about 20 lines of code) and does not support some of the enhanced features, e.g. caching or proxy servers, which would be required for a full fledged release. To allow for the later subsequent of these features, a `t_HTMLPage` object was introduced to encapsulate their details.

4.2.5 Connecting to other applications

MRTSpace is designed as a stand-alone program that relies on helper applications to display WWWAnchors that pointed to anything but a VRML file. The most common situation where such helper applications are needed arises when the user selects a 3D object that is associated with a hyperlink that points to a HTML page. In this case MRTSpace calls an external helper application, e.g. Netscape or Mosaic, to display the HTML page.

An alternative is to reverse the roles and use a HTML browser as the main program and provide only a plug-in to display VRML pages. But this limits the range of possible experiments with virtual realities, because a plug-in gets information only about the current scene, thus preventing all actions that concern the interfaces between such scenes. This interfacing is still a major point of experiments, for example there are attempts to generate geometrical continuity across scenes avoiding the 'teleport' effect. But this can only be done when the process of loading a new scene is under the control of the VRML browser itself.

Another connection was made to MRTPen, a general purpose interface to the MRT. By handing over a scene to the MRTPen, it is possible to further process the scene, for example to generate a raytraced image.

4.3 Enhancing the user interface

4.3.1 Motivation

VRML is designed as a scene description language. However, many people expect VRML to evolve into a virtual reality description system. For such a development the following issues have to be addressed:

Lack of consistent behavior. The point about virtual reality is to present some kind of reality with an inherent logic of its own. Most approaches try to copy physical behavior, but some succeeded to build up their own logic. Yet the first generation VRML browsers did not implement any such logic at all. Although the scenes used several metaphors that implied a specific behavior, this behavior was not supported by the browser. For example, many scenes implemented buildings, but their inherent logic of separating outside from the inside was broken by the fact that a user could walk straight through a wall. This violation of the expected behavior resulted in the feeling of moving through a 'faked' world, where the visual reception was not matched by the behavior.

Lack of communication. The concept of cyberspace was based on the ability to communicate from the very beginning, but VRML was designed to support only a single user without any way to communicate with other users.

Lack of animation. Although VRML gives the designer several ways to define 3D objects, there is no way to animate them, therefore, VRML is not able to represent the dynamic character of many real world objects: doors can be opened, cars move around or elevators move up and down. To implement such features, the designer needs a language to describe animations depending upon the actions of the user, which exceeds the scope of VRML.

Lack of individualization. People have the inherent need to change the world they live in, either to customize it to their need or to express their personality. To those people who expect to live primarily in cyberspace such features are essential. But even the more mundane oriented missed this feature, because without it even a simple task like leaving a note for a friend becomes impossible.

The need to improve VRML was broadly accepted, but due to the nature of the VRML community, the emphasis which problems to concentrate on was biased. Those people originating in the world of 3D design opted for heavy additions to allow sophisticated animations, which led to the development of VRML 2.0 and ActiveVRML. On the other hand, those coming from the network design drove VRML towards a multi-user environment. Compared to these, the fields of consistent behavior and individualization received much less attention.

From these four themes, two were chosen to be further pursued: consistent behavior and multi-user support. The reason for this selection was that the versatility of the MRT promised to be an advantage in these fields. Chapter 6 will cover the multi-user support and this chapter will show a new approach to provide a more consistent handling of complex situations.

Although the designer of a virtual reality has a vast amount of freedom, there are two common approaches to define rules that describe the behavior of objects inside this world:

1. Matching the user's expectation

If the rules that describe the internal logic of a virtual world adhere to the expectations of its user, he will have little trouble to navigate this world. Therefore, many designers choose the common every day experience of the user to create associations between the 3D objects and their behavior. For example, a 3D objects that looks like an elevator is expected to provide

the functionality of a real world elevator. If the virtual reality matches this expectation, the user will be able to interact with it in a most natural way.

The set of metaphors that is available for this kind of virtual reality depends on the intended type of user. A virtual reality that has a world wide audience will have to be restricted to symbols known to everybody, whereas a virtual reality that is intended for a special cultural group can make intense use of those metaphors that are bound to this culture.

2. Creating new rules

On the other hand, there are common reasons to use objects with a behavior that deviates from the users expectation:

- A virtual reality that always meets the users expectation is boring. Although this may be the intention, there are several applications where the element of surprise is welcome, for example games or advertising.
- Although using objects with a functionality that is inherited from the real world, many virtual realities choose to avoid their limitation in time or space. For example an elevator in a virtual reality could use an arbitrarily high acceleration to avoid an annoying delay.
- Sometimes the cost of simulating the real world behavior is prohibitive or such a simulation is plainly impossible, so that an approximative behavior or even a substitution is implemented. A typical case of such a substitution is an optical bell.

Although such a concept has an artificial character in the beginning, some of them will reach a wide spread use and reach a state of common acknowledge. For example, the concept of a teleporter is accepted by a broad group of people.

For the designer of an interface to a virtual reality, the implementation of a well-chosen behavior of certain objects can significantly help the simplification of the user interface.

Most user interfaces let the user navigate a 3D space using physical input devices that provide two degrees of freedom. To navigate a 3D scene, at least three degrees of freedom are required to navigate a scene: turn left-right, turn up-down, move forward-backward. Therefore, most interactors provide some kind of multiplexing the two degrees of freedom provided by the input device to the three degrees of freedom needed for navigation. However, this kind of multiplexing is associated with the need for additional user training.

MRTSpace introduces another approach: instead of using multiplexing, the user is given only two degrees of freedom for movement: rotate left-right and move forward-backward, which can intuitively be controlled with a 2D pointing device. To allow movement along the up-down direction, some objects are given properties which allow movement along the vertical axis. For example, a staircase is a very natural way to move up and down. The rules that describe such a behavior will be described in detail in chapter [4.3.2](#).

Most of the rules implementing such behavior must typically be applied each time the user moves or performs an action. Therefore, the algorithms implementing these rules have to be fast enough to be executed in real time. This requirement depends on many variables: the available computation power, available memory, required display size and the rendering algorithm, which itself depends on many more variables. Furthermore, most of these variables are likely to change within the next years, thus every attempt to give a more precise definition of the term 'real time' will be difficult.

Nevertheless, we can at least give reason for the following upper bound: Algorithms that require time superlinear in the number of polygons in the scene should not be used to provide

advanced object properties. Assuming that the time to render a picture is linear in the number of polygons - which is true for most z-buffer based rendering engines - any algorithm that takes more than linear time will predominate the rendering process for large numbers of vertices.

This upper limit strongly restricts the selection of algorithms to implement collision detection, which is essential to simulate real world behavior in a virtual world. To implement a general collision detection between a moving object and a static scene, the moving object is extruded along its movement path and is then intersected with the static scene. But a general intersection of this is associated with high computation times, which exceed the constraint of linear computation time by far [5].

There are several approaches to reduce the number of polygon intersections to be performed by structuring the objects either as octree [34] or in volume hierarchy [13]. Moreover there are several approaches that trade precision for speed. For example, it is possible to construct a distance matrix between the objects to be tested for collision. This is done either by using raytracing techniques or Z-Buffer algorithms. The complexity of this type of algorithm is linear in the number of polygons involved [30]. On some platforms, the Z-Buffer based algorithm can even be implemented using the rendering hardware [21]. Although the complexity of these algorithms grows only linear to the number of polygons, their implementation multiplies the time to display a frame by a factor that is too high to allow a real time display on the intended platforms. Moreover, the algorithms shown in the next chapter introduce the concept of user guidance, which requires several intersection tests per frame.

Therefore, MRTSpace uses the ray casting ability of the MRT to approximate collision detection. This approach is very fast, but has a very bad resolution. Nevertheless, this low resolution proved to be high enough to implement most of the rules simulating real world behavior.

A strong point of the MRT that allows a very flexible usage of ray casting is that a single object format is used for all the different rendering models like approximation, ray tracing or radiosity. This reduces the implementation to a few lines of C++ code. A simple collision detection that sends a ray ahead of the avatar can be done with a single line of code:

```
int hit=scene->
intersect(avatarPosition,avatarDirection,NULL,stepWidth,hitObject);
if (hit)
{
    // handle the collision
}
```

`scene` is an object that represents a collection of 3D objects and `intersect` tests whether a given ray (originating from `avatarPosition` heading into `avatarDirection` with a length of `stepWidth`) hits any object inside this scene.

4.3.2 The heuristic rules

The MRTSpace supports a set of rules for implementing a behavior that matches the expectations that users might have. Most of these rules can be described on an abstract level, for example: 'always stay upright' or 'When hitting a wall, turn away from it'. To support such rules, the browser needs to extract additional information from the scene, either to classify a situation or to find an appropriate reaction. This is done by casting rays, which act as a kind of sensors.

By convention, the unit length for cartesian 3D coordinates is assumed to be 1 meter.

1. Guessing the avatar's size & auto selecting start mode

A problem with using VRML as a basis for a virtual reality system is that it does not include any means of describing the avatar. Therefore, the size of the avatar is unknown and has to be guessed by the browser. One heuristic is to assume that the initial camera position is in a distance above the ground that is equal to the height of the avatar. This makes sense for most scenes that are built to be navigated in a walk mode, but is not appropriate for scenes that are intended for the fly mode, where the camera is most times placed outside the scene, to allow an initial overview.

So, whenever a user enters a scene, one ray is cast down from the camera position toward the ground. If the ray hits an object, the distance to this object is assumed to be the avatar's height (see Fig. 4.2) and the browser starts in the walk mode. Otherwise, a size of 1.8m is assumed and the browser starts in the fly mode. If these rules produce an undesired result, the user can always alter the mode by buttons, or changing the size of the avatar. This size is used as a basic measure of how fast the avatar should move around in the walk or fly mode.

2. Detecting walls & obstacles

The most confusing phenomenon that is encountered by unexperienced users with a typical browser is that walls are not solid. So it may happen that a user who just wanted to move towards a wall walks through the wall, ending up at an unexpected position. The worst case is that the user left the walkable area of the scene, and drops into the big void that surrounds every VRML scene. But even if the user manages to stop in front of a wall, he still is often confronted with the 'white wall syndrome', which describes the problem that a user who comes too near to a wall sees only a single colored screen and has no hints for orientation. Texture mapped walls give a much better information for orientation, even if the user comes very close.

MRTSpace therefore has the following rule: if the user moves too close towards an object, he will be stopped, thus preventing the user from walking through the wall. This will be done at some distance, called the head distance which is one tenth of the avatar's size (see Fig. 4.2). To avoid the 'white wall syndrome', the avatar is turned away from the wall if the user insists on going ahead, until the avatar faces a direction parallel to the wall and may continue to go ahead. This, together with the rule that if the user hits a wall at exactly rectangular angle to turn right, allows to explore many scenes by just going ahead and decide only at branches where to go.

Whenever the user tries to move ahead, a ray is cast originating at the current position and aimed toward the destination position. If this ray hits an object at a distance less than the distance to be moved, the avatar hit that object. If the distance is greater than the head distance, which is a tenth of the avatar's size, the avatar may travel only up to the head distance towards the object in this frame. If this distance is less or equal to the head distance, the avatar will not move at all, but is turned away from the obstacle (see Fig. 4.3). To decide whether to turn left or right, the normal of the point of intersection is compared to the direction of the avatar:

```
if (dot(avatarDirection * surfaceNormal, avatarUpVec-
tor) <= 0)
    turn right
else
    turn left
```

`surfaceNormal` is the normal of object hit by the ray at the point of intersection.

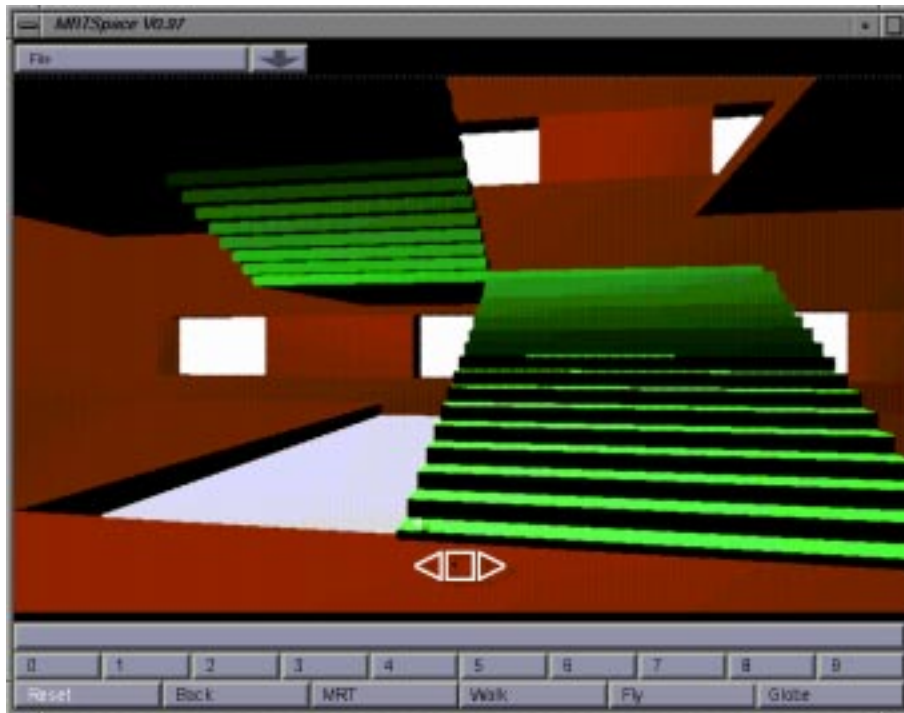


Figure 4.1: The Interactor object as used by MRTSpace.

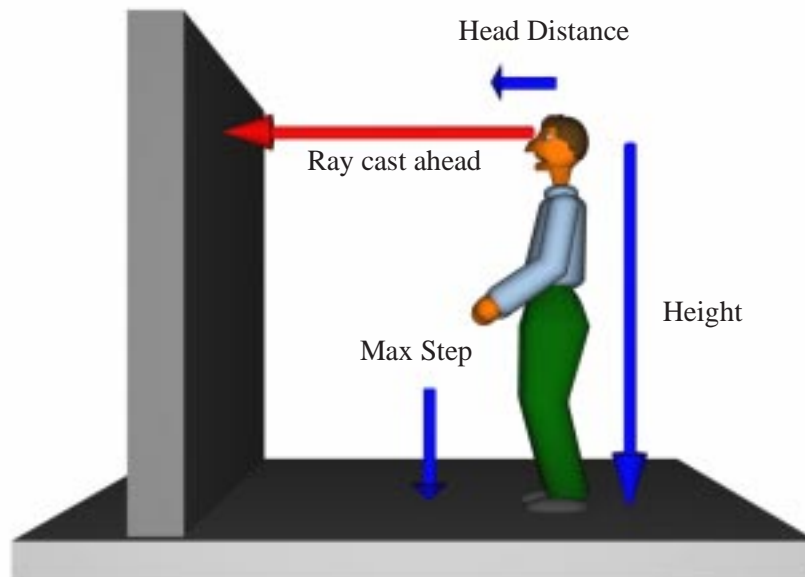


Figure 4.2: The basic magnitudes in determining the behavior of the avatar.

3. Climbing, Falling and landing

One of the ironies of VRML is, that although introducing a 3D view, the scenes developed radically towards flat 2D groundplans. The main reason for this are the difficulties many people encounter when using the fly-mode. Therefore, the MRTSpace introduces the concept of climbing to allow the navigation of 3D ground plans with the easy to use walk mode. Given the concept of casting rays to measure distances, the implementation was simple: one ray is cast down at the current position and one ray is cast down from the destination position. The difference between these distances is then compared against a maximum step height (most times 1/5th of the avatar's height, see Fig. 4.2). If this difference is less than the step height, this difference is added to the up-axis of the avatar. A typical situation is a user who moves forward a staircase (see Fig. 4.4). If the current position is in front of the staircase and the destination position is on the first step of the staircase, then the avatar will not only go ahead but also go up, to match the natural process of climbing a stair.

If the climbing height is greater than the step height, the object is considered to be an obstacle (see Fig. 4.5) and handled accordingly. This way the collision detection is greatly enhanced, some objects that will not be detected by the ray cast ahead, will be detected by the ray cast down. A typical example is a desk that is easily missed by horizontally cast rays, but not by vertically cast rays.

If the difference is negative and the user tries to descend a distance greater than the step height, the user will either fall down or go to the flight mode. The maximum falling distance is defined to be twice the height of the avatar. If the avatar falls down, several intermediate frames are generated, to give the user the impression of falling down (see Fig. 4.6). If the difference exceeds this falling height, the browser is automatically switched to the fly mode, giving the user the chance to go down in a more controlled manner. A typical situation where this kind of automatic switch occurs is a user who stands on top of a building and goes over the edge (see Fig. 4.7). Instead of falling down several stories, which may be confusing, the user will float in the air, and the transition from walk mode to fly mode is indicated to the user. If this was not the desired behavior, he may just fly back and will be automatically be switched back to the walk mode.

This automatic switch from fly to walk mode is called 'landing' and occurs under several conditions:

- The avatar's altitude drops below the landing limit, which is 1.1 of the avatar's height.
- The user is within the 'crash landing distance' of the ground(see Fig. 4.8). This is typically half the avatar's height. If this happens, the avatar's position will be updated to the full avatar's height above the ground (see below for a definition of 'ground').

With these rules for automatic switches between walk and fly mode, the user rarely has to use the explicit change state buttons, and most switches are performed just by using the movement controls.

4. Staying upright

VRML has no concept of 'up' or 'ground', therefore heuristics have to provide this information. The trivial solution is to take the vector (0,1,0) as a constant up direction, but this prevents the design of several interesting worlds. For example, a big sphere as the ground could not be walked with this kind of concept. Therefore, the MRTSpace enhanced the walk mode by forcing the up vector of the avatar to match always the normal of the ground the

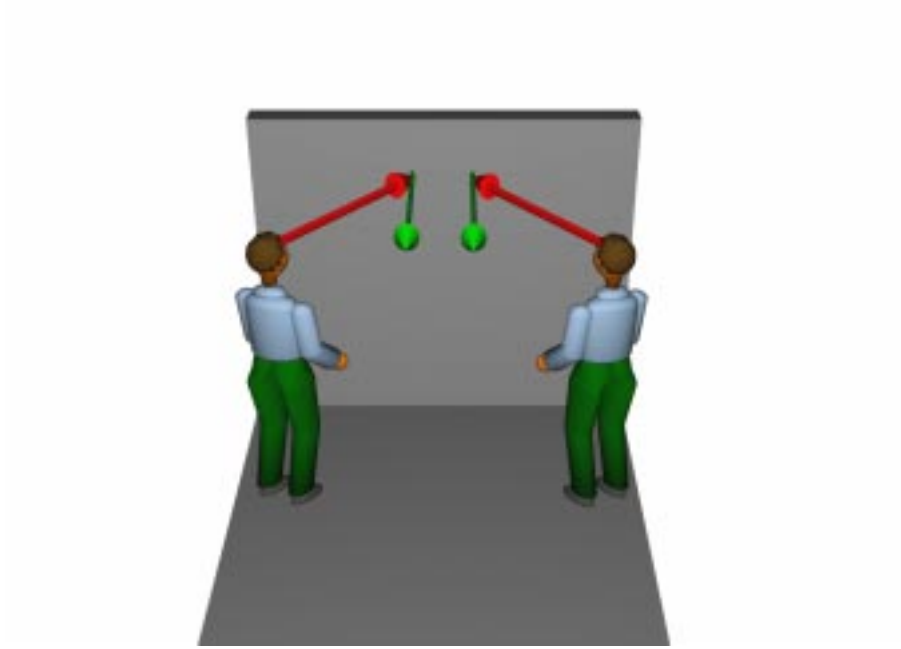


Figure 4.3: Turning away from a wall to avoid getting stuck.

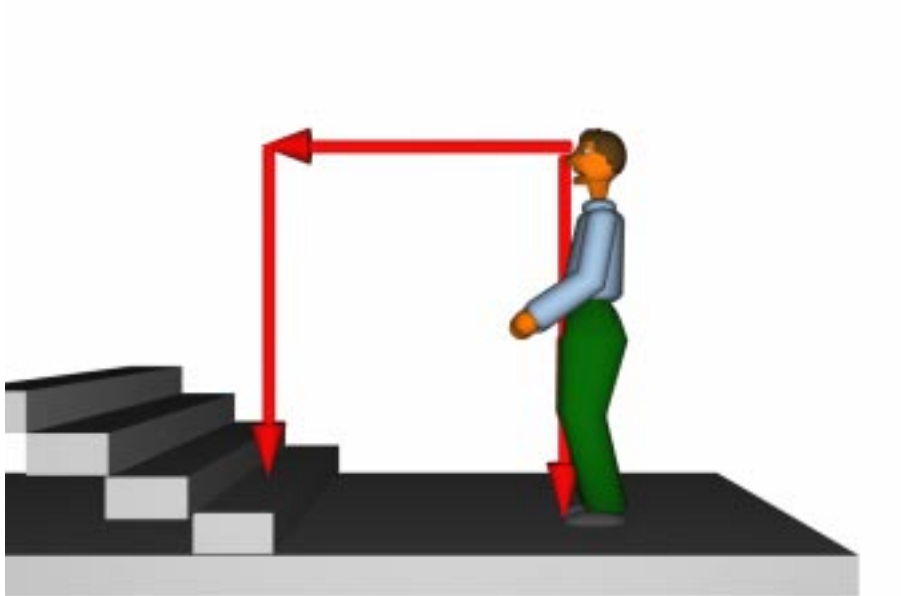


Figure 4.4: Detecting steps by casting rays downward from the current and the next position of the avatar.

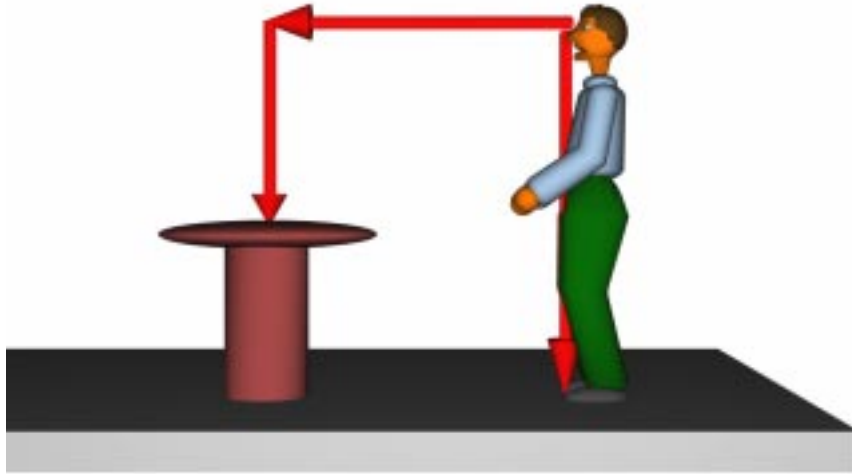


Figure 4.5: Detecting an obstacle by casting rays downward from the current and the next position of the avatar.

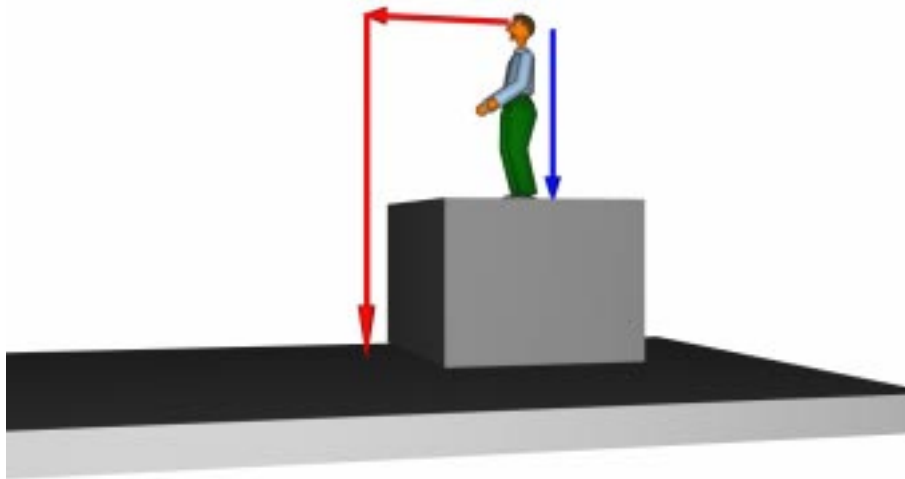


Figure 4.6: Detecting a situation where the avatar falls down by casting rays downward from the current and the next position of the avatar.

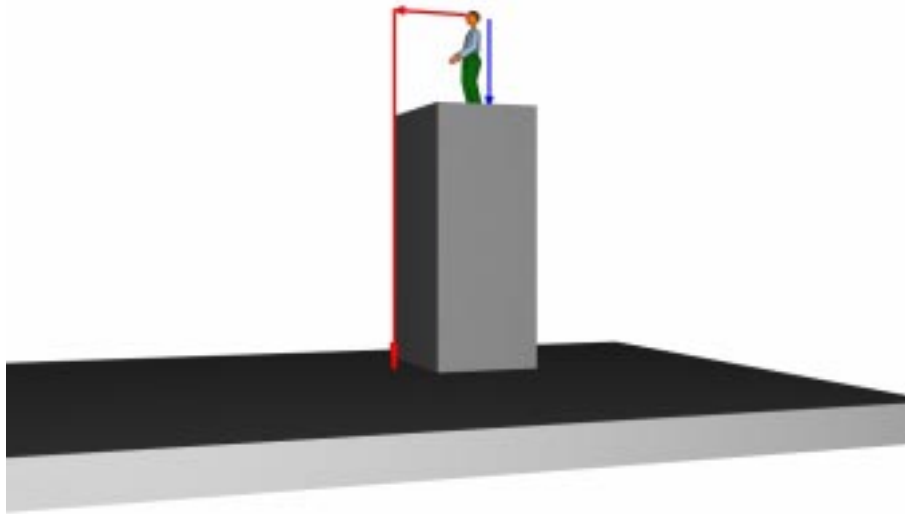


Figure 4.7: Detecting a situation where the browser automatically switches to fly mode by casting rays downward from the current and the next position of the avatar.

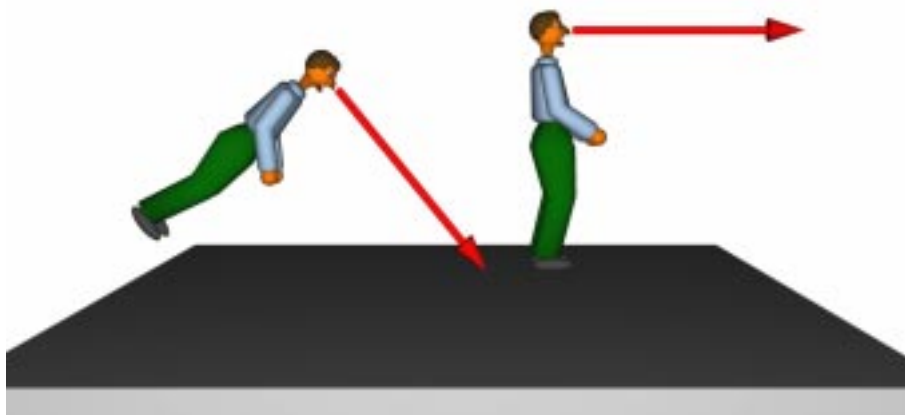


Figure 4.8: An avatar immediately before and after a crash landing.

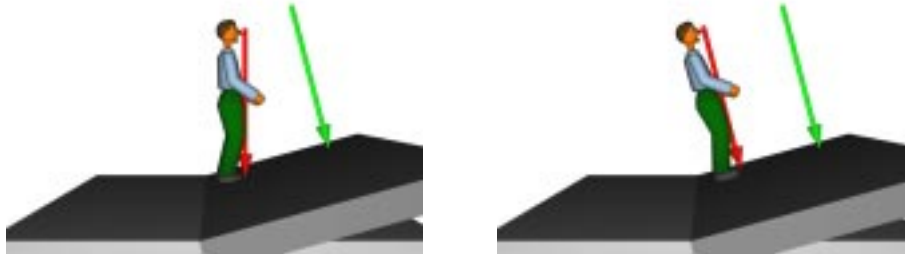


Figure 4.9: Adjusting the up vector of the avatar to the normal of the walking plane.

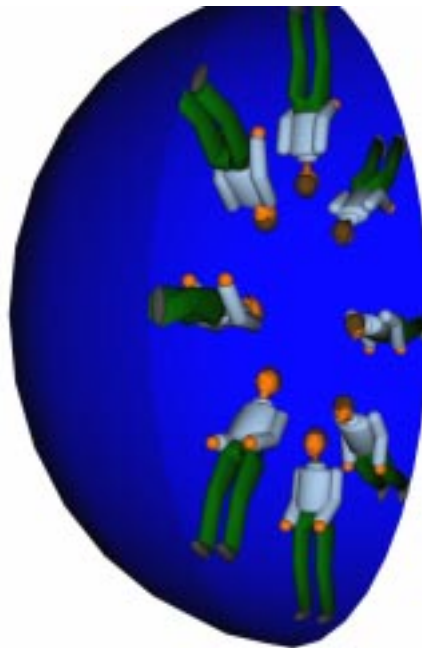


Figure 4.10: Walking inside a sphere.

avatar stands on (see Fig. 4.9). Thus, when walking on a sphere surface, the up vector of the avatar will match the radial vector after each step. An even more interesting example is to walk the inside of a sphere(see Fig. 4.10): If a world is built on the inside of a sphere, it is very easy to get a complete overview, just by looking up and around. Flying from one point inside a sphere towards another is much simpler than doing this on the outside of the sphere.

Because of this concept, there are two types of faces that are considered to be ground in the fly mode: every face with a normal within some angle of the current up vector this allows the user to land on any surface in a controlled manner or every face with a normal within some angle of the last up vector of the walk mode, which is mostly used to detect the crash landing of a user who lost the sense of orientation in the fly mode.

5. Globe mode

The globe mode, sometimes called trackball mode, is mainly intended to examine a VRML scene that contains just a single object and not a walkable world. Most VRML browsers take the center of the scene as the center of rotation, but MRTSpace extends this concept to be usable inside complete worlds. This is done by selecting the center of rotation at a point of interest in front of the user. For example, if the avatar is inside a virtual museum, positioned in front of an artefact, and the user switches to the globe mode, several rays are cast to search for the object to be examined in front of the user. If such an object is found, its center will be the center of rotation and the user may examine the object easily from all sides.

To find the object of interest, a fixed array of rays is cast from the avatar's position into directions similar to the current looking direction (see Fig. 4.11). This array is built to provide a high density at the center of the screen, assuming that the object will be in the middle of the screen, and a low density at the outside (Gaussian bell curve). Then all rays are weighed and the ray with the best score is taken to be the center of the object. The base value for a ray is $\text{obj_distance} * \text{bell}(\text{center_dist})$, where obj_distance is the distance travelled before hitting an object, center_dist is the angle between this ray and the looking direction, and Bell is the function describing the Gaussian bell curve. To avoid centering on the floor, all rays that hit the ground are entirely removed from the competition.

4.3.3 Porting to other 3D packages

The main advantage of the MRT/VRML over other 3D packages is the ability of the MRT to cast rays, which provides a fast and convenient way to measure distances and to obtain the normals of certain faces of interest. To my very surprise there exists a sort of 'hack' that provides the functionality of ray casting with a typical 3D engine. Instead of casting a ray, the complete scene is rendered into an off-screen buffer, with the camera at the position of the ray source and looking into the direction of the ray. To obtain only the distance to the first hit object it is sufficient to render a scene with the size of one pixel and then query the z-value of this pixel from the Z-buffer. Most 3D rendering engines have this kind of low level call to obtain the z-value of any rendered pixel, but some lack the ability to render into an off-screen buffer. If both the distance and the normal are required, a 2×2 pixel scene is needed to calculate the normal by the differences of the z-values. With a typical 32-bit z-buffer, the resolution of the normal is at least high enough to support the algorithms of the previous chapter. A more sophisticated approach uses a 3×3 pixel scene, which gives a higher probability to detect several special cases, e.g. hitting an edge.

Another way to increase the resolution is to render the scene from a point very near to the hit point of the ray. In this way, a higher precision can be reached at the cost of calculation time.

4.4 Conclusion

The main point of the enhanced user interface of the MRTSpace was not to simplify the handling of existing scenes, but to extend the set of scenes that can be handled by the user in an intuitive way. A user examining a virtual world without such an enhanced user interface is constantly confronted with a situation where the expectation created by the optical reception of the scene is not matched by an according behavior. The very first VRML files provided a rich set of objects that were created with their real world counterpart in mind, but as the visitor of such worlds became frustrated by the mismatch of look and behavior, the designers avoided any usage of a metaphor that could lead to the expectation of any behavior. In this way, a lot of powerful metaphors were avoided, leading to a flat and unfamiliar world design. With the enhanced user interface, several of these metaphors are available again, and many more are possibly emerging from the creative usage of the underlying rules. Although these rules were first implemented using the raytracing ability of the MRT, they are portable to most 3D rendering packages. The next step is to merge these rules with one of the successors of VRML 1.0, which provide the ability to animate objects. Together these technologies come near to the promise that was given by the very first experiments with VRML: To build a Cyberspace [25].

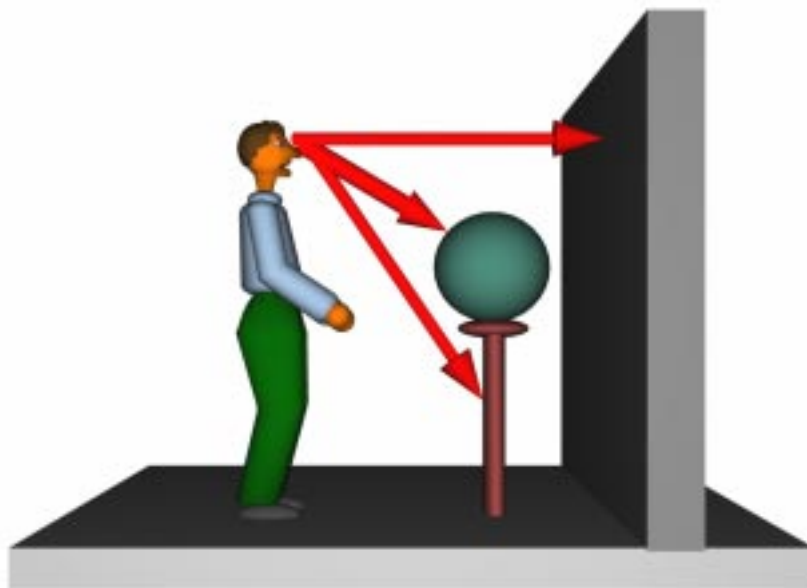


Figure 4.11: Using raycasting to find a center for the globe mode.

Chapter 5

IRC

The IRC is the Internet Relay Chat protocol (RFC 1459) [14, 29], which provides a rich set of methods to support text based conferences. It was originally developed as an extension to a bbs network, but was soon ported to the Internet. As the term 'chat' suggests, its primary intention was to provide a means for social contact, technical aspects of collaborating are secondary for the IRC concept. The most important feature of IRC is that it allows people to talk in near-to-real time with each other.

5.1 Users

IRC was designed to be available for any user who has access to the Internet. Although there are several client programs to make the handling of IRC easy, a telnet is all that is required to participate. Moreover, IRC is intended to be a public service, with very low access restrictions. There is no need to register in a permanent way to use IRC. In the original proposal it was recommended to use a verification mechanism to proof the real name and address of a user, but these efforts have been cancelled. Today, the real name of a user can be very easily faked, but as this fact is known to most people who are using IRC, the usage of faked names is accepted, sometimes even appreciated.

This role-play character of IRC is enhanced by the fact that every user has to assume a nickname, which is used to reference him in the discussion. For historical reasons, this nickname is restricted to 9 letters. As this nickname must be unique, there is a special protocol that resolves the collision of nicknames.

5.2 Channels

The term 'channel' corresponds to a group of users who want to chat with each other. In contrast to most mailbox systems, IRC has no fixed set of channels. Instead, every user can create a channel in a very easy way. When creating a channel, the user will be asked for a title, which should give some hint about the topic that is to be discussed on this channel. Every user can get a list of all available channels and join up to 10 of them, although chatting on several channels simultaneously is considered to be an advanced task.

5.3 Channel Operators

A user who created a channel is called the channel operator. He has some kind of responsibility and got some extra rights. For example, he may exclude a user who failed to conform to his expectation of social behavior. There are no means of controlling this power, but unsocial behavior of channel operators is very fast punished by the typical IRC user, who will just leave such a channel, create a new channel, and ask all people of the old channel to join him on the new channel. Such social algorithms proved to work quite well.

A special channel mode that allows to keep a channel free of unwelcome guests is the 'invited only' mode. In this mode, only people explicitly invited by the channel operator can join this channel.

Another important role of the channel operator is that of a moderator. In a moderated channel, everybody has the right to join, but only the moderator and the persons invited by him have the right to speak.

5.4 Servers

IRC is based on a client-server architecture. The client connects to a server – which permanently runs the IRC server program – using a standard TCP connection. The server will respond to each of the client commands in an asynchronous way with a short numerical reply. If the command requires further feedback, there is no means to associate the requests with their answers, but their ordering.

IRC can be set up to run only on a local network, but it was designed from the very beginning to run on a network of interconnected IRC servers, called the IRC backbone. These servers have a special protocol which allows the unification of all their individual user and channel lists. If two IRC servers are joining their name list and a duplicate nickname occurs, one of the users is required to change his nickname. If duplicate channels occur, these channels are joined, and whatever one user posts in a channel will be visible to users connected to any server. After establishing a connection between two servers and the initial synchronisation of their databases, the servers will constantly exchange messages to keep in synchronisation. Furthermore, there is a server-to-server message that propagates the content of each channel to any server with users who joined this channel.

5.5 Operators

There is a special class of users called operators. The task of these users is to maintain the technical and social integrity of the IRC network. There are several commands only available for a user who is in the operator state, for example connecting or disconnecting IRC servers. The most controversial right of an operator is to expunge a user from a server. In contrast to the right of channel operator to expunge a user from a channel, where the user can join other channels or create his own, an expunge from the server prevents a user from accessing any features of IRC. Moreover, any operator has the right to expunge any user from any server. As the status of operator is often granted to any system administrator of an IRC server, this is one of the weak points of IRC. To prevent abuse in the long run, the fact that a user is expunged will be forwarded to any user who was on the same channel as the expunged user.

5.6 Efficiency

Besides their management functionality, the IRC backbone is also used to lower the net load. In a system without such a backbone each time a user speaks to the channel – the most common message form – the client would have to generate one message addressed to each of the other users on this channel. When a user whose client is connected to the IRC backbone speaks to a channel, the client sends a single message addressed to the channel to the IRC server. In the trivial case all recipients of this message are connected to this server, which will send one copy of the message to each of them. If one or more users of this channel are connected to another IRC server, the IRC server connected to the speaker will send a single message addressed to the channel to the IRC server connected with the recipients. The receiving IRC server will send one copy of this message to each connected user who is in the addressed channel. By delaying the copying of the message, the netload associated to this message will be minimal as long as it travels only inside the IRC backbone. As the IRC backbone covers most of the long distance lines, this reduces the netload for connections where the bandwidth is most precious.

Chapter 6

Joining IRC with MRT/VRML

6.1 Motivation

A point of rapid development is the extension of VRML towards the support of multi-user virtual realities. The ultimate goal is to build a world wide cyberspace, which allows people all around the world to share a common virtual reality. To cope with such an intended scale, there are several problems to be solved. This work will focus on two of the central questions:

1. How do the participants exchange information? There are basically two different communications protocols in use: unicasting or multicasting.
2. Where is the definition of the virtual world stored? There are two ways to represent the dynamic state of the world: either in a central server or distributed over several peers.

In this thesis, new approaches to solving these two problems are introduced: It proposes IRC as a communication protocol, because the performance of IRC comes near to that of multicasting, but in contrast to multicasting, IRC is available from any internet connection. Moreover, this works chooses a distributed representation of the virtual world, because this approach lowers the demands on the server down to a level that can be handled by existing protocols like HTTP or FTP. Therefore, the time-consuming task of verifying and installing a new protocol on a server is avoided.

6.2 Unicasting vs. Multicasting

When implementing a multi-user virtual world, there are several situations where a program has to send several copies of a single message to several recipients: Whenever an avatar enters or leaves a world, moves, talks or changes its shape, this information must be propagated to all clients that are within the range of perception of this avatar. There are two basic techniques to handle such situations:

6.2.1 Unicasting

If a participant needs to inform others of some event, this event is sent to each recipient in a directly addressed message. Therefore, the netload imposed by a single event is directly proportional to the number of recipients. As the number of events increases proportional to the number of interacting users, the overall netload will rise to the square of the number of interacting users.

Although the unicasting approach is very easy to implement, this square complexity is a serious drawback. The main method to lower the netload in an environment based on unicasting is to lower the number of people that can interact with each other. This can be done by limiting the range of vision or hearing or by dividing the virtual world into smaller parts, which do not allow interaction across their borders. Nonetheless, these measures can only reduce the factor, but the square complexity itself remains.

6.2.2 Multicasting

The internet protocol has a special type of addressing called multicasting [28], which allows the addressing of a group of recipients, thus a single message can be received by many computers. Therefore, the cost to propagate an event to a group of participants of a multi-user virtual world is constant and the overall netload rises only linear to the number of interacting users.

A typical multicasting usage follows these steps:

1. All participants must agree on a common multicasting address. As a multicast has potentially world wide range, a multicasting address must not be used by any other group in the Internet. There is a central registration of all multicasting addresses, which allows an organisation to claim exclusive usage of a constant multicasting address ¹. Moreover, there are several protocols which allow the assignment of a dynamic multicasting address on demand.
2. All recipients must tell this multicasting address to their underlying operating system. From now on, the operating system will listen for multicast messages and forward those that match the given address to the corresponding application.
3. A multicast message can be sent just like any message via the user datagram protocol (UDP), which is one of the five protocols used by the Internet. Only the address marks this message as a multicast message. The upper 4 bits of the address are 1110, which is a unique marker for multicast messages.
4. Every application that registered this multicasting address in their operating system receives all multicast data packets to this address just like any other data packet of the user datagram protocol.

The main advantage of multicasting is that by transmitting a data packet once, several recipients can be addressed. Without multicasting it would be necessary to send this data multiple times, once for each recipient. But there are some complications associated with multicasting:

1. Only very few computers have hardware support for multicast messages. Without hardware support, every time a computer receives a multicast message, the cpu must be interrupted and a cpu process has to be activated to decide whether the incoming packet is to be forwarded to an application. This may result in a heavy cpu load, even if none of the multicast messages is of interest for any application running on this machine.
2. A normal router cannot decide whether to forward a multicast message to another network or not. But if all routers forwarded all multicast messages, any single multicast message would be transmitted to all computers connected to the Internet, thus imposing an extreme netload. So most routers just filter out all multicast messages, thus limiting the range of a

¹the first organisation with such a claim was SGI, who called their address SGI-Dogfight.

multicast message in a typical environment to all computers that are on the same physical network cable (Even most hubs filter out multicast messages).

The underlying problem is that a multicasting address contains no information about the receivers of this message and therefore all the standard routing algorithms fail. There is an attempt to overcome this deficiency of multicasting: The IGMP [Reference: RFC1112] protocol allows a computer to inform all routers that are on the same physical network which multicast messages are to be forwarded. This information will be forwarded to all other routers which may need it to forward a multicast message to the requesting computer. There is an experimental network of such collaborating routers called the 'mbone' [28], which stands for multicast backbone, but only very few computers have access to this network. One simple reason is that the netload associated with the mbone is quite high, as typical applications include real time TV and audio. Therefore, a dedicated 1Mbit/sec connection to the mbone is considered to be the minimum.

3. Because of multicast messages being transferred with user datagrams, they are not reliable. To provide a reliable communication, the participants must implement some high level protocol.

Despite the current limitations of multicasting, it is reasonable to argue that the multicast approach is superior to unicasting and that most problems associated with the mbone can be solved by either a denser net of mbone servers or a more sophisticated ethernet hardware.

6.2.3 Using IRC as a distribution protocol

Although the IRC is entirely based on unicasting, it provides the functionality of multicasting to the application: a single message to the IRC server is propagated to a group of recipients. The performance of IRC depends on the distance between each of the participants and the IRC servers.

The worst case is a situation where all participants are on the same physical backbone and the IRC server is far away. The netload and response time will both rise to the square of the number of interacting users. Moreover, the travel time to the next IRC server will increase the response time by a constant factor.

The best case is a situation where the distance to the next IRC server is small compared to the overall distance to be traveled and each client connects to a different IRC server. Here the netload is typically less than square to the number of interacting users, because the IRC backbone can bundle messages along long distances. More importantly, the response time will be independent of the number of recipients.

To understand the implications of this best- and worst-case behavior, it is essential to consider the available net bandwidth in each situation: The worst case occurs if all participants are on the same local network, which typically provides a very high bandwidth. The best case occurs often in a situation where the participants are spread all over the world and the net bandwidth for such long distance connections is poor. Therefore, the square complexity is associated with a high net bandwidth and the linear complexity is matched by a poor bandwidth, resulting in a very well balanced overall performance.

A disadvantage of IRC in its current form is a 'flood control' that was implemented by some servers to prevent a user from annoying other users by sending messages at such a high rate that a channel becomes unreadable. This flood control prevents users to exceed an average of one message each two seconds, although it allows up to 5 messages to be sent within a very short time.

Another remarkable point is that IRC is currently based on unicasting, because multicasting is not widely available, but as soon as the availability of multicasting is high enough, IRC will be

able to adapt it as an alternative technique to send a single message to several clients on a single network. The form of the basic communication messages is well prepared for this step.

6.3 Central vs. Distributed representation

6.3.1 Central representation

One common approach to implement a multi-user virtual world is to dedicate a server to store and maintain the representation of the virtual world. Whenever a user enters the virtual world, his avatar is integrated into this central representation. Whenever the avatar performs an action, the central server updates its representation of the VR accordingly. Every change to the central representation is forwarded to all clients that may be affected by the change. Because of this central representation, conflicts between users competing for a resource are easy to solve. Moreover well known algorithms of client-server architecture can be applied. Although easy to implement, there are some disadvantages of the central approach:

1. A special protocol needs to be installed

To implement a world-wide shared virtual reality, it is necessary to provide a high number of servers that provide the required protocols. But the installation of a new protocol, which has to run in most cases in a privileged mode, always is a security risk.

2. The netload is not well balanced

As each of the clients needs to communicate with the server each time some change of the virtual world occurs, the number of messages to be processed by the server is much higher than that of each client. In a small local network, this is of no concern, but a server that operates world-wide will need an extremely high bandwidth.

3. High CPU load for the server

Compared with other protocols like HTTP or FTP, the cpu load imposed by a virtual reality server is high. Especially the routing of messages and the conflict resolution inevitably will impose a high requirement for cpu power. This need cannot be satisfied by the current backbone of servers that form the internet. Most of these servers have a high storage capacity and data throughput but a low cpu power, because this matches the classical requirement for a mail-, news- or ftp-server.

4. The world does not fit into a single server

One central server that manages the entire Cyberspace would radically simplify many problems, but it is doubtful whether this is ever technically possible². However, when distributing the Cyberspace over several servers, several problems – especially conflict resolution – are much harder to solve and many advantages of the central server approach are reduced.

6.3.2 Distributed representation

An alternative to the centralized server approach is to distribute the representation of the virtual world over several machines, which allows to distribute the netload and cpu load.

²Even CompuServe, who tried to do this for quite a long time, has learned that a cluster is the only way to handle a world wide communication concept.

To allow for such a distribution, the representation and description of the VR is divided into a static and a dynamic part. The static part describes objects which do not change under any condition, such as walls. The description of static objects may include a set of states that the object can be in, e.g. a door might be defined as being either in the state 'open' or in the state 'closed'. For each 'multistate object' its initial state must be specified in the static description. (Thus, the entire static part of the VR describes a multistate object and its initial state.) The dynamic part specifies the actual state of the multistate objects described in the static part, and objects added to the VR by users. The actual state of multistate objects may be changed by user actions, e.g. a user might open a closed door. Typical examples for user added objects are user-defined avatar shapes (which may take the form of arbitrary VRML URLs).

After dividing the representation into a static and a dynamic part, the static part is stored in a server, which is now a very simple task, and the representation of the dynamic part is shared by all clients. In its most simple form, all clients have a complete representation of the dynamic information, and one of them will forward this information to any new client. In an extended approach, each client holds at first only the information needed to display the current surrounding of the avatar. If the avatar moves to another section, its client must query the other clients for the dynamic information in this section. Therefore, each dynamic piece of information must be stored in at least one client. The canonical way to distribute this dynamic part along the clients is: Every client stores all states that it has changed and all user defined objects he has created. Moreover, only the last client that changed an object has to store the state of this object.

The distributed representation has a major drawback: when a user leaves, all changes he has done will be no longer available to the other clients. A more sophisticated approach may redistribute the information stored in the leaving client to the remaining clients, but with this approach, the description of the actual state of the VR is lost when the last client terminates. For some applications like a café or a conference room this limitation may be acceptable, but other applications like a public blackboard rely on the persistency of their objects.

Moreover, this approach has an increased netload compared to that of a central representation, because there are typically several clients answering a query for the dynamic state in a certain area. Although the total amount of transferred information is the same as with a central server, the number of messages will increase and each of them has a certain overhead imposed by the network header or the minimal block size. The other side of this increased netload is that it is distributed over several computers. Assuming that each client has an equal share of the dynamic information, the netload will be perfectly balanced, no single computer will have a higher bandwidth requirement than its peers. The only exception is the login process, where the single server has to provide the static definition of the world but even this can be distributed: Because the server only has to provide a static file, there can be several servers that mirror the original definition.

Another advantage of this approach is that the server is only required to store a static definition of the world, which is a quite simple task. There are several existing protocols, like HTTP or FTP, which can provide this functionality. Therefore, a virtual world could be created without the need to install additional protocols. Developing and establishing new protocols is very time-consuming. This is especially true if these new protocols require privileged access to system resources, as security issues have to be considered very carefully in this case.

6.3.3 The commercial aspect

Besides any technical questions, the choice between a central and a distributed representation has a social dimension. There are three major commercial providers of multi-user virtual worlds:

- Fujitsu: WorldsAway³
- Microsoft: V-Chat⁴
- Time Warner: Palace⁵

They all prefer the central distribution for the following reasons:

1. A central distribution requires a central login, which is a good time to charge money.
2. A central server makes it easy to exclude people who do not fit into the social pattern targeted by the provider.
3. A central server makes it possible to enforce social rules and punish unwanted behavior, e.g. harassment.
4. The well known techniques of client-server architectures are applicable leading to a stably running program.
5. Although the initial costs to provide the required cpu power and net bandwidth are high, this entry cost will keep the number of competitors low.

The reasons why many non-commercial organisations or single individuals prefer the distributed approach are:

1. There is no need for an expensive central server, because the cpu and net load will be distributed shared among all participants.
2. There is no need to install special programs on the server, only the clients will have to update their applications. This will significantly speed up the distribution and evolution of the virtual reality.
3. Because the required algorithms are not well known, this is an interesting field for the community of computer science.

But in most cases it is a single argument that forces people to choose one or the other approach: A non-commercial organisation just does not have the money to provide a computer that can handle several hundred users in real time. A commercial organisation just needs some kind of central control over the virtual reality.

6.4 Common requirements

The following three sections show three approaches that differ in whether to use unicasting or multicasting and whether to use a central or a distributed representation. To simplify their comparison, the following list of questions will be answered by each approach:

³<http://endeavor.fujitsu.co.jp/hypertext/news/1996/Feb/28-e.html>

⁴<http://www.msn.com/v-chat/default.htm>

⁵<http://www.thepalace.com/home.html>

1. How does a new user connect to the virtual reality and how does the client get the initial description of the virtual world?

There must be some initial protocol to get in touch with the virtual reality. After establishing this initial connection, the new client must get a description of the actual state of the world. For a static world, this can be done by downloading a constant world description. For dynamic worlds, the state of all objects that can be altered also has to be transferred. Because all the avatars can be considered to be part of the dynamic world, this information may be transmitted with the overall world description, but most protocols handle the avatars in a special way, because the avatars have special information attached, e.g. their name or e-mail address.

2. How are the actions of an avatar distributed?

Whenever an avatar moves or if it changes its appearance, all other clients within some range must be notified.

3. How do users communicate with each other and is there a way to provide privacy?

A user may talk to a single other user in a private manner, or he may choose to address all other participants within a certain range. This problem is similar to the problem of the distribution of the avatar's action, but a finer control of the audience is required. Typical forms are:

Private one to one a private message is sent from one user to another, who is the only recipient.

Public one to one a message is addressed to a single person, but any other user who is nearby will be able to receive it.

One to group a single person talks to a group of users, typically all who share a single room.

Public group in a kind of forum, the right to speak is restricted to a fixed group and others may only listen.

Private group a group of person agreed to speak in private, no other user may speak or listen to this group.

When the Cyberspace is thought to be a central meeting point for people, there also must be a way for a small group to detach from the publicity to talk in private.

4. How are conflicts resolved?

When the user is allowed to change some part of a dynamic world, there arises the need to resolve conflicts. One type of conflict occurs when two different users try to set one parameter of the world at the same moment to two different values. Another type of conflict arises when two users compete for a resource that cannot be shared. An example for this kind of resource is the space occupied by a single avatar or the right to speak in a forum.

6.5 The server centered approach

The first approaches to build multi-user virtual worlds were based on a classical client-server architecture, using only unicasting and a central representation of the virtual world.

The basic idea of a server centered approach is to put all information in a central place and to distribute just as much information as needed by each client. All clients send their actions to the server, which updates its representation of the virtual world. Whenever such a change is of interest for a client, the server will send a unicast message to this client. Even client to client communication is routed via the server, allowing for a very fine control of who is addressed.

In its pure form, every action of a client has to be acknowledged by the server before its effect becomes visible to the user, but as the typical response time for a world wide connection is much higher than the acceptable response delay for a real time application, most clients cache some part of the virtual reality. Every action is first applied on the local copy of the representation of the virtual world, allowing a very fast responsiveness. Although this technique is essential to build a multi-user virtual reality with the existing bandwidth, it may lead to a state where several clients have different or even contradictory models of the virtual reality. This may lead to the annoying effect that a user performs an action based on his reception of the world, but this action is invalid with regard to the state of the world stored on the server. In this case, the server needs to have a method to find some way to inform the user that his browser was out of synchronisation and his action was refused.

The following list answers the basic list of questions arising in a multi-user virtual world based on a server centered system:

1. How does a new user connect to the virtual reality and how does he get the initial description of the virtual world?

The client connects to a single and constant IP address. After the login is verified, the client will receive some partial description of the virtual world and a list of all other participants that are within some range.

2. How are the actions of an avatar distributed?

The server has a database which keeps track of all avatars and their actions. All clients mirror this central data base. If an avatar performs an action, the client immediately updates its database accordingly. Moreover, these changes are posted to the server, which in turn will send update messages to all clients that may be affected.

3. How do users communicate with each other and is there a way to provide privacy?

Whenever a user talks to other visitors, this is handled like any other action and the client will send a message to the server, which will propagate this message to the addressed clients. For a central server that routes all the user to user messages, it is very easy to provide a very fine control of who receives which message.

4. How are conflicts resolved?

Most multi-user worlds use some kind of speculative execution for the clients to reduce the typical feed back time. But this speculative execution may fail whenever two users compete for a resource that cannot be shared. After detecting such a situation, the server must choose one of the competing users to get access to the resource. The user who is denied access will be notified, which may result in the confusing experience that the last actions of him are undone. Although the algorithms needed to solve such conflicts are not simple, their implementation is simplified by the fact that a single instance exists which can decide who of the competing users wins. The algorithms to achieve this in a distributed system are much more complex.

6.6 The multicasting based approach

There are several approaches (for example [18] that are based on multicasting and a central representation of the virtual world. Only the initial setup of a new client will be performed using unicast messages. After this initial setup, the client has an exact mirror of the description stored in the central server. All following messages are sent as multicast messages, either originating in the server or any client. Each of these messages posts a change to the state of the virtual world. These changes will be applied to any locally stored description of the world and to that in the server to keep all these definitions in synchronization. If for any reason it happens that two clients have different representations of the virtual world, it is the task of the server to resolve such a conflict. Moreover, the server must also resolve any conflicts that may arise when two users compete for a resource that cannot be shared.

The following list answers the basic list of questions arising in a multi-user virtual world based on multicasting:

1. How does a new user connect to the virtual reality and how does he get the initial description of the virtual world?

After login, the server tells the client logging in the multi-cast address which will be used to interchange information about any actions of the participants and the point-to-point communication. Moreover, the client gets a complete description including the position and description of all other avatars. This is the last time of an individual point-to-point communication between the client and the server, all other messages will be exchanged using the multicasting protocol.

2. How are the actions of an avatar distributed?

Each client whose user performs an action sends a multicast message which is received by all other participants and by the server. Each of them will update the internal representation of the world. As the multicasting protocol is based upon user datagrams, which are unreliable, there must be some high level protocol that ensures that each of the clients representation of the virtual world matches that of the server. Using multicast messages imposes a netload which will grow linear to the number of participants, which is much better than the square netload imposed by the server centered approach. Moreover, the lower netload results in much better synchronization and response time.

3. How do users communicate with each other and is there a way to provide privacy?

User communication is handled just like any other user action and multicasting is used to distribute it. Although privacy can be provided by a high level protocol, it is very easy to break, because multicast message are very easy to intercept for anybody who has access to the mbone. So, the only way to provide more than a polite request not to interfere is to use encryption.

4. How are conflicts resolved?

The conflict resolution for conflict by users competing for some resource that cannot be shared is handled by the server, just like it is done in the server centered approach. The only difference is that a speculative execution of user action that did fail will influence all clients, therefore they will be visible more times than in the server centered approach.

6.7 The distributed approach

The basic idea proposed in this work is to use a distributed representation of the virtual world combined with IRC as the communication medium. As the underlying language is VRML 1.0, which was designed to describe only static worlds, the only dynamic objects are the avatars. This dynamic information will be stored in each of the clients, leaving only a static definition of the world to be stored by the server, a simple task that can be performed by any HTTP server. To interchange the changes of the current state of the world, the clients use an IRC channel that is dedicated for this purpose. To link an IRC channel to the VRML file, there is a new VRML-node containing the name of that channel.

The main advantage of this approach is that no new protocol is needed, which was a requirement of traditional multi-user worlds. Moreover, IRC is – in contrast to multicasting – available to every internet user.

The following list answers the basic list of questions arising in a multi-user virtual world based on a distributed approach:

1. How does a new user connect to the virtual reality and how does he get the initial description of the virtual world?

The static part of the world is stored as a plain VRML page on any HTTP server. The only change is an additional node, which contains the name of an associated IRC channel. As the static information never changes, there may be an arbitrary number of copies, each of them pointing to the same IRC channel.

After reading the static definition of the virtual world, the client will join the corresponding IRC channel. From now on, IRC provides the functionality of a multicasting connection: A single message to the IRC server will be propagated to any other client who joined this channel.

After joining a channel, the client sends the definition of its avatar to all other clients. Each of them will respond with a complete definition of their corresponding avatar. In this way, a new client receives the dynamic information about this world a short time after joining a channel.

2. How are the actions of an avatar distributed?

The clients will send IRC messages to each other to propagate their actions and keep all the representations of the virtual world in synchronization.

3. How do users communicate with each other and is there a way to provide privacy?

User-to-user messages are sent as standard IRC messages and can make use of all IRC features, including several ways to provide privacy. There is a private point-to-point connection or the possibility to open a channel which can be joined only by invited guests. To differentiate actions from user-to-user messages, action messages are labelled with a special tag.

4. How are conflicts resolved?

The current implementation has no resources that can not be shared. The collision detection is always performed on the local representation of the world, and some trivial algorithms are used to avoid dead locks.

The new VRML node that links this VRML file to an IRC channel has the following form:

```
SharedRoom
  width    2  # SFFloat
  height   2  # SFFloat
  depth    2  # SFFloat
  name     "" # SFString
```

The first three fields describe a bounding volume. If these fields are empty, the given IRC channel in the 4th field is assumed to be used regardless of the position of the user. If a bounding volume is given, the associated IRC channel will be used whenever the avatar is within this bounding box. When bounding boxes are nested, the user will receive all messages on any level, but he will transmit only to the channel associated with the innermost bounding box. This allows to implement a hierarchical structure for user communication and interaction. For example, a virtual cafe could have a single IRC channel attached to each table, thus avoiding a too noisy background.

As IRC is used to transport both the user-to-user messages and the actions of avatars via a single channel, the messages containing avatar actions are marked by starting with a '!'. If the user enters a message starting with a '! ' it should be sent as '!!', with the 2nd '!' removed by the receiver. It is allowed to send multiple vrml messages within one IRC message, each starting with a '! ', although the maximum length of IRC messages of 511 bytes must not be exceeded.

The following special messages are implemented:

Sign on message !000 <Avatar definition>

This message is sent by any user joining a multi-user virtual reality. The avatar definition may contain any valid VRML node. This will be most probably be a 'Separator' which contains a list of objects. An alternative is to send a 'WWWInclude' which contains the URL of a VRML file containing the definition of the avatar's shape. There are two advantages of this approach: A conflict with the maximum message length of IRC can be avoided and the VRML browser can implement a cache for the most common avatars, thus reducing the netload.

There is no need to send a starting position or orientation because they are determined by the initial camera value in the VRML file. Some additional information about the new participant can be gained via the IRC protocol, including the e-mail address.

Avatar definition !001 <Avatar definition>

This is the first response of any participant to a sign-on message, containing the definition of the own avatar.

Position definition !002 <transformation node>

This is the second response of any participant to a sign on message, containing the position and orientation of the user's avatar.

Typically, both the avatar definition and the position definition will fit into a single message, but an avatar definition may take up all the space provided by a single message, and the position definition will be sent in its own IRC message.

Avatar redefinition !003 <Avatar definition>

The user may choose to change his avatar's look to express some kind of minimal gesture. A special feature of the VRML browser is to allow the DEF/USE mechanism within an avatar definition to avoid any resending of avatar shapes. This requires that the VRML browser keeps a separate name space for each of the participants. If the browser does not support this feature, any avatar definition but the first one containing DEF or USE must be ignored.

Position redefinition !004 <transformation node>

Whenever the avatar moves or turns, a corresponding transformation node will be sent. It is up to the VRML browser to prevent the icon from popping around by interpolating between the successive positions.

Log out message !005

This message requests all other participants to remove this avatar.

Because IRC requires a nickname as an address in private one-to-one messages, MRTSpace uses a hashing algorithm to generate an eight-byte string from the users e-mail address. If this nickname is already in use, it is incremented by 1 and then tried again. To allow a simple distinction between users on IRC using a VRML based browser and those of classical IRC browsers, nicknames generated by MRTSpace start with a '#'.

To incorporate the IRC into the MRTSpace, a general multi-user communication object is implemented which hides the actual underlying communication technique. This encapsulation will both ease future experiments with other communication protocols and make it possible for other applications to use IRC by using this object. The base object is named `t_MultiComm` and has a child called `t_MultiCommIRC`, which implements all the required IRC communication methods.

The `t_MultiComm` is interwoven with the internal dispatcher and the interactor. At initialization time, a child process is created, which will wait for any message received on the socket that is connected to the next IRC server. Whenever this child receives a message, it pipes this message to the parent process and posts a user event to interrupt the interactor, which will react to this user event by returning control to the internal dispatcher. The internal dispatcher will then call the `t_MultiComm` object to disassemble and interpret the IRC messages. The `t_MultiComm` uses the `t_VRMLParser` to parse any message containing VRML objects. If any avatar changes its position, orientation or shape the interactor is informed to update the scene currently displayed.

6.8 Conclusion

Before the introduction of IRC as a communication protocol, the decision whether to use a server centered or a multicasting based approach was quite simple: The server centered approach promises to be stable and highly available, whereupon the multicasting based approach provides the highest performance. IRC is in most respects a compromise between these two approaches, at best it combines the performance of multicasting with the stability and availability of the server centered approach, but in the worst case, its performance may be degraded down to that of the server centered approach. The strength of IRC is that its best case is associated with a situation where the bandwidth is usually low, and its worst case with a situation that provides a high bandwidth. This leads to an overall performance that is very well balanced, and makes best use of scarce resources. When comparing IRC with multicasting in a situation where the participants are spread all over the world, the connection to the respective backbones becomes less important and the performance of the backbone will be the main factor. At this level, the algorithms used by the IRC backbone and mbone are very similar, both try to find the optimal routing path and both can delay or bundle packages to increase performance. But in contrast to the mbone, the IRC can be accessed by any Internet client using only unicast messages. Both the IRC backbone and the mbone are expected to grow at a significant rate, which may be even accelerated as these backbones are used to build a shared virtual world.

Another interesting argument to use IRC to build virtual worlds is that it makes more sense to add 3D functionality to an already existing communication technique rather than to re-invent this communication technique as an add-on to a 3D browser. Moreover, this argument can be extended

to question a current trend which requires to speed up the frequency at which new standards and protocols are released, often at the cost of their quality. It may be a rewarding approach to take the rich set of protocols already available by the Internet and use these protocols to implement new applications, rather than to match each new idea by a new protocol. The MRTSpace is an example, where a new technology – a multi-user virtual world – could be implemented using only existing protocols like HTTP and IRC.

Chapter 7

Conclusion

This thesis proposes three ideas which have the potential to bring the current development of VRML a step further towards the implementation of a true Cyberspace.

The first idea addresses the problem that VRML user interfaces are not satisfactory because objects have no simulated physical properties. Unfortunately, a complete physical simulation is prohibitively expensive computationally. However, some key physical features of objects can be modeled using only algorithms with a complexity linear in the complexity of the scene. MRTSpace uses the raycasting functionality of the MRT to implement a set of heuristic rules governing interactions between avatars and VRML objects. These rules prevent users from walking through objects (or getting too close to objects). Furthermore, the rules detect the plane the avatar walks on and automatically adjust the vertical avatar position. As a result, users can explore VRML scenes like way they would explore new environments in the physical reality. The users themselves are relieved from having to take care to conform to the physics implied by the VRML objects, e.g. they can walk up stairs without having to adjust the vertical coordinate manually, and they can follow walls without having to watch out not to inadvertently walk through them. Thus, the user interface provided by MRTSpace can be used very intuitively.

The second idea is to distribute the dynamic part of the description of a virtual world among the clients. Only the static part of the description of the virtual world is stored on a central server in the form of a VRML file. This approach avoids any concentration of network and cpu load anywhere in the network, loads are well balanced and thus, the system scales well with the number of users connected to the virtual reality.

The third idea is to use IRC to overcome the lack of multicast support by the Internet in its current state. IRC is entirely unicast based on the TCP/IP layer, but it implements multicast functionality on the application layer (in order to provide conference / chatting functionality). By using IRC to propagate information describing the dynamic processes in the virtual reality, it is possible to implement a VR system with a multicast architecture that allows remote users in today's Internet to connect to a common virtual reality. Multicasting architectures are a key base technology for building efficient VR architectures.

These ideas have been implemented in the MRTSpace to provide a proof of concept. MRTSpace is one of the first applications allowing extensive user interaction that was developed on the basis of the MRT and PENGUIN. To better support the implementation of such an application, the API of the MRT was redesigned to be completely object-oriented.

There are two promising options for the further developmet of MRTSpace. The first is its extension towards a complete IRC client. Although MRTSpace uses IRC as a communication protocol, it does not support all the features provided by a typical text based IRC client. If MRTSpace

included the few missing functions, it could become an interesting alternative to the classical IRC client programs.

The other direction the future development of MRTSpace may take is its extension towards VRML 2.0, which would add animations and interactions to the now inanimate worlds of VRML 1.0. Such a combination of interactive objects with a multiuser environment would provide the basic technologies to meet the expectations that are associated with the term 'Cyberspace'.

Bibliography

- [1] Ames, A. 1996. *VRML 2.0 Sourcebook*. John Wiley & Sons, Incorporated. ISBN 0-471-16507-7.
- [2] Ames, A., Nadeau, D., Moreland, J. 1995. *The VRML Sourcebook*. John Wiley & Sons, Incorporated. ISBN 0-471-14159-3. 11
- [3] Bell, G., Parisi, A., Pesce, M. 1995. *The Virtual Reality Modeling Language – Version 1.0 Specification* <http://webspacesgi.com/Archive/Spec1.0/index.html> | 11
- [4] Bell, G., Carey, R. Marrin, C. 1996. *The Virtual Reality Modeling Language Specification – Version 2.0* <http://webspacesgi.com/moving-worlds> | 11
- [5] Boyse, J.W. 1979. "Interference Detection Among Solids and Surfaces." *Commun. ACM* **22**: 3-9. 28
- [6] Donnelly, C., Stallman, R. 1995. *Bison: The YACC - Compatible Parser Generator*. Free Software Foundation. ISBN 1-882114-45-0. 16
- [7] Fellner, D.W.. 1996. "Design of a Graphics Architecture Bridging the Gap between Modeling and Rendering." Presented at: *The Fourth International Conference in Central Europe on Computer Graphics and Visualization '96*, Feb. 12-16., Plzen, Czech Republic. 5, 7
- [8] Fellner, D.W. 1996. *MRT – A Visualization Tool Addressing Problems ‘outside’ the Classical Rendering Domain* VISMATH Workshop, Berlin, May 30 - June 3, Fischer 1995
- [9] Fellner, D.W., Fischer, M. 1996. "Computer Graphics Interface (CGI) - a Good Concept and a Valuable Tool for Research and Teaching in Computer Graphics" *Computers & Graphics* **20(2)**: 341-346 10
- [10] Fellner, D.W., Fischer, M., Weber, J. 1993. *CGI-3D – A 3D Graphics Interface*. Tech. Rep. IAI-TR-95-x, University of Bonn, Dept. of Computer Science, Bonn, Germany. 10
- [11] Fischer, M. 1995. *PENGUIN – A Portable Environment for a Graphical User Interface*. Tech. Rep. IAI-TR-x, University of Bonn, Dept. of Computer Science, Bonn, Germany. 10
- [12] Frécon, E., Hagsand, O. 1995. *VRML 1.0 Syntax Checker*. ftp://ftp.sics.se/pub/dive/vrml_parser.tar.gz |. 23
- [13] Hahn, J.K. 1988. "Realistic Animation of Rigid Bodies." *ACM Computer Graphics Proceedings SigGraph '88* 22:299-308. 28
- [14] Harris, S. 1995 *The IRC Survival Guide*. Addison-Wesley ISBN 0-201-41000-1. 6, 39

- [15] Hartman, J., Wernecke, S. 1996. *The VRML Handbook*. Addison-Wesley Publishing Company, Incorporated ISBN 0-614-14427-2. 5
- [16] Jucknath, O. 1996 *MRTSpace – Multi-User 3D Environment using VRML* accepted to Web-Net'96 San Francisco
- [17] Levine, J., Mason, T., Brown, D. 1992. *Lex & Yacc*. O'Reilly & Associates Inc. US ISBN 1-56592-000-7.
- [18] Michael R. Macedonia, Michael J. Zyda, David R. Pratt, Donald P. Brutzman, and Paul T. Barham 1995. *Exploiting Reality with Multicast Groups* IEEE Computer Graphics and Applications Vol. 15, No. 5 50
- [19] Neider, J., Davis, T., Woo, M. 1993. *OpenGL Programming Guide*. Addison-Wesley ISBN 0-201-63274-8 19
- [20] New Riders Publishing Staff. 1996. *Java Professional Reference*. New Riders Publishing. ISBN 1-56205-598-4. 15
- [21] Okunev, O., Kunii, T. 1995. "Fast Collision Detection." *The Visual Computer* 11/95: 497-511. 28
- [22] OpenGL Architecture Review Staff. 1992. *OpenGL Reference Manual*. Addison-Wesley ISBN 0-201-63276-4
- [23] OpenInventor Architecture Group Staff. 1995. *OpenInventor C Plus Plus Reference Manual*. Addison-Wesley ISBN 0-201-62493-1 5
- [24] Pesce, M.D. 1995. *The VRML Programming Library Version 1.0*. <http://vrml.wired.com/vrml.tech/qv.html> | 23
- [25] Pesce, M. 1995. *VRML Browsing & Building Cyberspace*. New Riders Publishing US ISBN 0-56205-498-8. 37
- [26] Raggett, D. 1995. *Definitive Guide to HTML 3.0*. Addison-Wesley ISBN 0-201-87693-0 5
- [27] Ready, K. 1996. *Webmaster's HTML & WWW File Format Reference*. New Riders Publishing ISBN 1-56205-617-4 5
- [28] Savetz, K. 1996. *MBone: Multicasting Tomorrow's Internet*. IDG Books Worldwide. ISBN 1-56884-723-8 43, 44
- [29] Shefski, B. 1995. *Interactive Internet*. Prima Publishing US ISBN 1-55958-748-2. 6, 39
- [30] Shinya, M., Forgue, M.C. 1991. "Interference Detection Through Rasterization." *Visualization Computer Animation* 2: 131-134. 28
- [31] Stevens, R. 1995. *Advanced Programming in the Unix Environment*. Addison-Wesley, Reading, MA. ISBN 0-201-56317-7 23
- [32] Wernecke, J. 1994. *Inventor Mentor* Addison-Wesley ISBN 0-201-62495-8 5
- [33] Witthaus, N. 1996. *Active VRML*. Prima Publishing. ISBN 0-7615-0742-6. 15

- [34] Yang, Y., Thalmann, N.M. 1993. "An Improved Algorithm For Collision Detection." *Pacific Graphics* **1**: 237-251. 28
- [35] Richter, J. 1995. *Advanced Windows: the developer's guide to the Win32 API for Windows NT and Windows 95* Microsoft Press, Redmont, WA. ISBN 1-55-615-677-4. 23