Collaborative object modelling in virtual environments

The development and evaluation of a multi-user modeller

Mårten Stenius

In partial fulfilment of the requirements for a masters's degree, at

School of Computer Science and Engineering

Royal Institute of Technology, KTH

Stockholm, Sweden

1995-1996

Fleranvändarmodellering av virtuella världar

Mårten Stenius

Examensarbete (20 poäng)

Datatekniklinjen

Kungliga Tekniska Högskolan

Stockholm

1995-1996

Abstract

In this report, the design, implementation, and evaluation of DIVEdit, a prototype collaborative object modeller for virtual environments, is described. Some preliminary user tests have been made in order to identify some questions and problems related to teamwork in virtual worlds.

In addition, an overview of some existing approaches to object modelling is given, some offer collaborative solutions while others make immersive shaping possible. DIVEdit combines some of these features, making an interactive, collaborative object modeller. DIVEdit is implemented as an application in the DIVE system for research on distributed virtual environments.

Sammanfattning

Denna rapport beskriver design, implementation och utvärdering av DIVEdit, en fleranvändarmodellerare för virtuella världar. För att identifiera frågor och problem rörande samarbete i virtuella världar har också några inledande praktiska försök utförts.

Vidare ges en översikt av några existerande system för objektmodellering. Både modellerare för virtuella världar och CAD-system för flera användare har studerats. I DIVEdit sammanförs valda egenskaper från dessa system till en interaktiv virtuell modellerare för flera användare. Modelleraren har utveckats för DIVE, ett forskningssystem för distribuerade virtuella världar.

Preface

This report is part of the requirements for a master's project in computer science for the author. The main part of the work was carried out 1995 at the Interaction and Presentation Laboratory (IPLab) of the Department of Numerical Analysis and Computing Science (NADA) at the Royal Institute of Technology (KTH), Stockholm, Sweden.

I wish to thank Kai-Mikael Jää-Aro for supervising me through this work and Yngve Sundblad for accepting me at the IPLab to do it, and everybody in the DCE group at SICS for helping me with DIVE. Thank you, Olle Sundblad, Hans Marmolin, John Bowers, my co-workers at IPLab and NADA, Anders Sandberg, Reine Bergström, Erland Lewin and others for valuable input (and Anders and Reine for also acting as test subjects) during the development and evaluation process.

And, most importantly, I wish to thank my family for all support, especially Karin for her love and patience which have helped me to complete this work.

Table of contents

Introduction

The goal of this master's project was to design, implement, and evaluate a collaborative object modeller for multi-user virtual environments. The modeller should be suitable for experiments on collaborative issues in virtual environments. Some preliminary user tests have also been made in order to identify some questions and problems that arise.

More generally, the purpose of this master’s project is to investigate some of the issues related to *collaboration* in virtual environments, and to outline how they may be supported. Current research on collaborative virtual environment involves, among many issues, work on *social interaction*, meaning supporting communication between the inhabitors, mutual awareness, appearance and so on (see for instance [Fahlén93]), or the sharing of *information spaces* (see for instance [Benford95]). This work is focused on *mutual modification* of a collaborative virtual environment.

Thus, a collaborative modeller was developed by the author, in the *DIVE* (Distributed Interactive Virtual Environment [Carlsson93; Hagsand96]) system for distributed collaborative virtual environments. The modeller allows the sketching of new objects and modification of the existing ones in any DIVE world. The design and implementation of this modeller, *DIVEdit*, is described in this thesis, as well as experiences from the practical use of it.

Finally, some issues that were found relevant to this work are discussed, such as:

- Subjective views
- Ambiguity in the representation of virtual objects
- Feedback in a multi-user virtual environment

Virtual world and object modelling

In this chapter, some examples of related research on object modellers and modelling techniques are described. There is of course a great multitude of applications to support object design today. From the most rudimentary—hacking coordinates with a text editor—to sophisticated CAD systems with precise control of angles, alignment, curves and so on. This is not intended to be a complete survey of current research and products available, but rather a selection which shows some interesting approaches.

To outline two ways of relating to the main purpose of this work—building and evaluating a multi-user object modeller in VR—I have split the examples into two main groups: *immersive modellers* and *collaborative CAD systems*. In addition, I round off this section with yet another examples of interest.

Immersive modellers

By an *immersive* application we mean that the user interface puts the user *inside* the virtual world, letting him or her interact within the world itself, while performing the task—in our case shaping three-dimensional objects.

Since research in this area is no longer a novelty, several modellers with varying degree of immersion have been implemented and studied. Three applications I have chosen to mention are *3-Draw*, *3DM*, and *JDCAD*—all with different approaches but with much in common.

Some things these examples have in common are:

- They use a perspective projection of the object (and world) being modeled. This means that the realistic feeling of objects getting smaller when at larger distance from the observer is preserved. However, it is less suitable if one wants to make measurements from the image: Angles are generally not displayed correctly, the apparent length of two objects vary depending on their distance from the viewer, and it is difficult to visually decide whether two objects are aligned or not. [Carlbom78; Foley90, pp 230-231]
- They use 6-DOF (degrees of freedom) input devices in various ways to interact with the 3D model.
- Although clearly related to CAD systems, their main use is fast sketching of objects.
- They are single-user applications.

The details are described below, along with reflections on what may be applicable to our situation.

3-Draw

The goal of *3-Draw* [Sachs91], developed at MIT, was “to develop a fundamentally new type of CAD system for designing shape”. The system is based on a two-part input device: in one hand, the user holds a palette (the “object sensor”) with which the wireframe model is rotated on the screen. The model is sketched in relation to this palette using a stylus, held in the other hand. The palette and the stylus are both tracked with 6-DOF Polhemus 3Space Trackers.

The idea is that the user should get a natural feeling of holding the object in his hands while shaping it: The object on the screen moves exactly as the user moves the palette. The stylus acts as a general input device for freely specifying, moving and modifying the curves that make up the model, as well as for controlling the program.

The approach was found to be very effective to quickly sketch relatively complex objects. This was partly attributed to the two-handed input technique [Buxton86], as well as the direct projection of the object on the screen.

3DM

3DM [Butterworth92], developed at the University of North Carolina, uses a VPL eyephone and Polhemus trackers for display and input. The user interface is “immersive”, with virtual toolboxes and menus, carrying over ideas from regular 2D drawing applications into 3D.

Other familiar techniques from 2D interaction used in 3DM are the bounding box selection of objects, rubberbanding, descriptive cursor shapes, and predictive highlighting (to highlight the item the user is currently pointing at before any selection is made, making the selection task easier for the user). Cut-and-paste operations were implemented, as well as an undo function.

The user may change his own size, and thereby make modifications on different scales. Objects are hierarchically grouped shapes, such as polygons, cubes, spheres and so on. However, there is no way of keeping two objects parallel. It was found that some of the problems rising from not having any constraints on the objects could be overcome by having a simple grid to snap the objects to. (For a discussion on a technique for snap-dragging objects in 3D, see [Bier90]).

3DM was a demonstration of what could be a tool both for inexperienced and sophisticated users, combining techniques from both traditional CAD and user-friendly drawing programs and moving into a 3D environment.

JDCAD

JDCAD [Liang93], developed at the University of Alberta, is a system for CAD prototyping using 6-DOF locator devices to track the user's head position and hand movements. The model is displayed on a regular computer screen, but updated according to the head position of the user to give the user some level of depth perception from the way the image perspective correlates to the position and orientation of the head (kinetic depth perception). The main interaction with the 3D interface is made by combining the tracking of hand movements (using an Isotrak 6-DOF tracker) with the pressing of different keys on the keyboard.

Among the techniques developed for JDCAD are:

- The *ring menu*, (Figure 1) which is activated by pressing a designated key. The user may then rotate the “bat” (an Isotrak 6-DOF input device) in his hand to choose between the menu alternatives which are ordered around a semi-transparent ring. An item is selected by rotating it into a gap in the ring (facing the user).

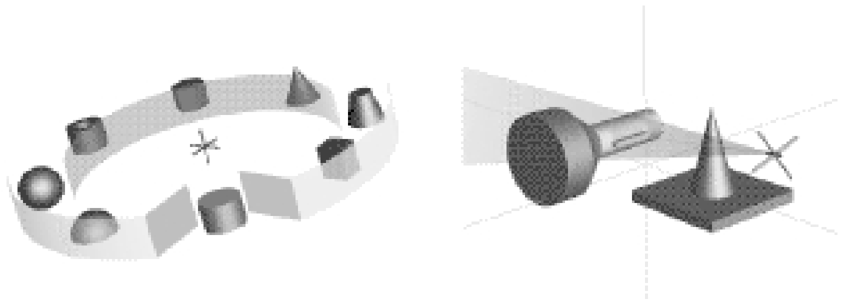

Figure 1: Techniques used in JDCAD: A ring menu and spotlight selection. Images from [Liang93].

- *Spotlight selection* (Figure 1) of objects enhances the classical laser-gun technique for pointing and selecting objects. Instead of just a ray extending from the cursor position, a cone is used. The purpose is to simplify selection of small and distant objects, which would be difficult to hit with just a ray. By giving objects higher priority the closer they are to the center axis of the cone, the accuracy for shorter distances is preserved. If several objects are hit by the spotlight, the one closest to the center ray of the cone and closest to the “bat” will be selected.
- The *bat-operated dial* is altered according to the rotation of the “bat” around its x-axis, a method found to be intuitive for specifying angles.

Noteworthy in the context of this work is also that the approach of reshaping with boxes around the selected object was found less effective for 3D than it is in 2D, noise in the input devices made it hard to pick the small handles (the technique used in 3DM, and also described in [Houde92]), and in addition they tended to clutter the environment visually. Instead, invisible “regions” around the object replaces the handles, and the user is informed of what handle is active by the shape of the cursor. For instance, when the cursor is moved close to the corners of a cube its *shape* would inform the user that the object would be resized if pulled there.

JDCAD also supports easy zooming in and out of the model, to make it easy to sketch fine details as well as the overall shape of an object.

JDCAD, in my opinion, is a very good example of how to exploit the possibilities of 3D interaction. However, since the usability of the techniques mentioned above relies on having a 6-DOF input device, they may be difficult to carry over to an application in DIVE, if we want to be able to use the simpler interaction techniques still supported by the system as well as the full-fledged HMD interface.

Conclusions

The 3-Draw project clearly shows how much may be gained simply by moving the interaction methods closer to the way we do things in “reality”. However, the devices were in this case specially shaped for the task of modelling objects—and may be less intuitive for other actions common in a more general virtual environment such as DIVE.

An important thing to learn from 3DM is that in designing applications for true 3D interfaces, you *may* benefit from what is being used in 2D design. Those techniques may gain a lot from moving into 3D, but still one should remember that they come from a user interface with more constraints, the largest one being the lack of a third dimension. While these restrictions may have been a problem in some cases, they may have been beneficial in other situations. As an example, consider a situation where the user wants to change the size of a box, but keep the size constant with respect to the *y* axis, using a “standard” bounding box with handles. With a 2D user interface, it would simply be a case of viewing the object as projected on the *xz* plane and drag the handles to the desired scale. But with a 3D modeller, there is “by default” too many degrees of freedom in movement—three, while we in this case only need two. Thus, some way of temporarily constraining the movement needs to be added, which of course is possible but adds to the complexity of the user interface.

Multi-user CAD

Multi-user CAD systems are interesting to me, since they support object modelling with more than one user, and in various ways show the usability of some of the collaborative features that are supplied in a distributed VR system such as DIVE. The most significant examples are collaborative pointers and graphical indicators of the viewpoints of the different users.

Two systems featuring collaborative CAD are mentioned below: *Co-CAD* and *Teledesign*.

Co-CAD

Co-CAD [Gisi94], developed at Hewlett-Packard and CNR-ITIA (National Research Council of Italy, Institute of Industrial Technologies and Automation), is a prototype CAD system supporting multiple users. It is designed to support both synchronous and asynchronous co-operation, as well as to function as a single-user CAD system. The effort to make a multi-user CAD system is motivated by the assumption that geographically spread-out design groups will become more frequent, which would create a need for distributed and collaborative applications within the area.

The group collaboration is supported by having a common database with the objects being edited and a possibility to have own views as well as to synchronise views depending on the situation. To support discussions around the design, a *shared pointer* is implemented—it may be moved by any user, but all users have to use the same pointer. No support for voice communication is implemented; conference telephone calls have been used instead.

Experiments with multiple users showed that some kind of owner control of objects was necessary. Accordingly, a three-level permission system, analogous to the standard UNIX file permission system was implemented which by default let the creator of an object be the owner.

It became clear that the working process of the users, mechanical engineers designing objects, does not motivate a constant collaboration through Co-CAD. Rather, the collaborative feature is intended to be used for quick consultation or discussion around a design made in private.

Teledesign

Teledesign [Shu94], is a CAD system for several users, developed at the Computer-Aided Design Laboratory at MIT. Since the system was specifically developed to identify and investigate groupware-related topics in CAD, the work is relevant in the context of collaborative virtual modelling.

Teledesign develops the idea of a “telepointer” further. Here “viewpoints” are supplied, which give visual information on the *viewing angles* of the different users directly in the display of the modelled object. It was found that a natural feeling of the “positions” (or viewing angles) of the other users clearly is a big help when discussing and shaping objects. (Taking the idea of “viewpoints” even further leads to the “avatars”, virtual bodies, that have become the standard metaphor in systems for multi-user virtual environments. See for instance [Fahlén93].)

Conclusions

Both Co-CAD and Teledesign approaches the subject studied in this report from a different, but not unexpected, angle. They take the more or less “classical” way of doing CAD, and add the possibility of multiple users onto it.

The result is interesting in that we see how metaphors are developed that can be compared to the user interaction model found in DIVE and other systems for multi-user virtual environments. The visual viewpoint indicators in Teledesign could be viewed as a very rudimentary reminiscent of a body icon, or “avatar”, as described in the next section of this report. Compare the telepointer of Teledesign to the interaction ray used in DIVE to indicate the interaction focus of a user.

An alternative approach

An interesting example of a 2D modeller for 3D worlds is described below. It combines a pretty straightforward window-based interface with the creation of 3D environments that doesn't depend too much on the third dimension when it comes to overall planning.

VR Mog

VR Mog [Colebourne96], developed at Lancaster University, is a floor-plan modeller intended to produce DIVE worlds (Figure 2). With a regular two-dimensional window-oriented interface, the user is able to plan the world from a “birdseye view” by positioning and scaling predefined “elementary” shapes (typically imported from other modelling programs) in the window.

To me, this approach seems very useful in many situations, especially for general world planning, when one wants to have a good overview. The user interface may seem simple—but it is clear and appropriate, and does not involve any unnecessary features.

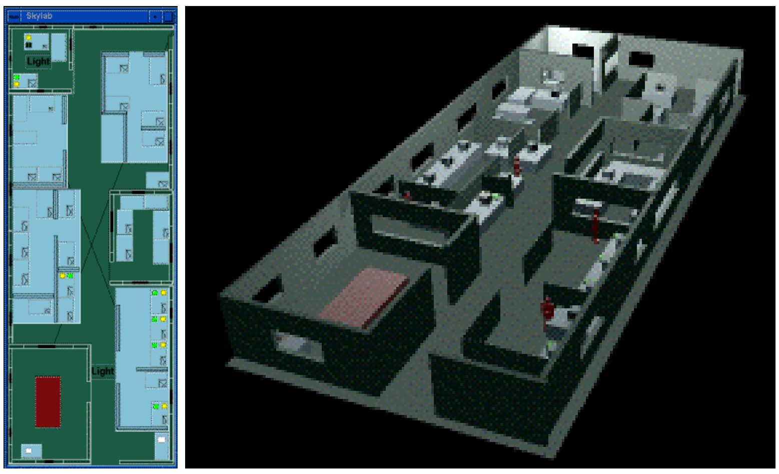

Figure 2: The “Skylab” world, created with VR Mog. Image from [Colebourne96].

I think a very important lesson is, that it is often quite enough with two dimensions. It *might* even be that many modelling applications, like world planning, are more difficult with an immersive, fully 3D, modeller. Provided of course that the overall structure of the world in question is primarily 2D, as is the case in for instance office or town planning. Think of how architects and engineers have been working for centuries—apparently the method works when the “2D modeller” consists of pen, ruler and paper, so why should it not work for virtual constructions? The human perception is (by nature or experience?) tuned towards navigating and orienting in basically two dimensions—the ground plane. But on the other hand, it could perhaps be argued that the physical restrictions that force us to move in “two” dimensions (the ground plane) do not exist in a virtual world unless explicitly defined. Practical daily experience with virtual environments shows, though, that a world without some kind of “ground plane” reference easily becomes disorienting—this is easy to verify with any 3D browser at hand.

Another interesting note regarding VR Mog is that it is clearly oriented towards designing *environments*, rather than *objects within environments*. This actually turned out to be a relevant problem in the practical experiences with the DIVEdit modeller described in coming chapters of this report.

Some more reflections

From Teledesign and CoCAD we learn the importance of awareness of other users in the context of modelling. Such functions are readily supported in the DIVE system, a question is whether they are sufficient for our task, or if they need elaboration.

In modellers such as 3DM and JDCAD we see examples of three-dimensional user interface widgets, such as the virtual toolbox and special cursors. In general, though it is doubtful whether such techniques are necessary at a first stage, when the key point of this work is *collaboration* in virtual environments, not user interface—even though lessons about the latter may come from the former.

DIVE

DIVE [Carlsson93; Hagsand96; DIVE96], is a distributed multi-user virtual reality system, developed at the Swedish Institute of Computer Science (SICS).

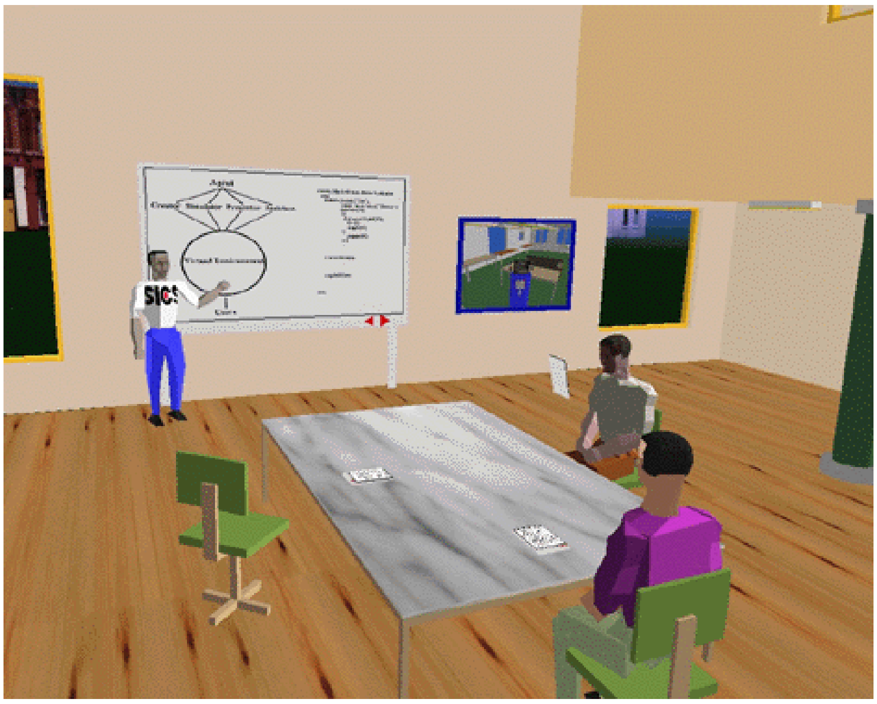

Figure 3: A DIVE conference. Image from [DIVE96].

DIVE has been in development since 1991, and was then part of the MultiG program, which sought to examine applications of high-capacity networks (with more than 1 gigabit/second capacity, thus the name) [Carlsson92]. The main feature of DIVE (initially named *TelePresence*) was from the start the support for several users, and this still holds today. The simple graphics of the first versions of the system has evolved and today DIVE supports features such as textured worlds with animations, video input, sound, active objects and integration with the World Wide Web.

DIVE is continuously being used to examine a wide range of aspects of shared, distributed virtual environments; from the fundamental network level, over the design of different applications and interaction methods, to collaborative and social issues. A typical DIVE application is the conference scenario, depicted in Figure 3, where a group of users may meet, interact, and share virtual documents.

Central concepts in DIVE

DIVE is based on a distributed database, which means that each participating user process holds its own local copy of the world description and that updates to the world are continuously sent over the network. The distribution is built upon the SICS Distribution Package (SID) [Hagsand95:1].

The basic element in this database is the *entity*, which may be of several basic types as described below. Entities are grouped into completely separated worlds, which in turn are accessed by the user processes.

Entities, objects

The concept of entities (or *objects*) in DIVE is very broad: Worlds are treated as objects, as are processes (at least their representations), light sources, and “regular” visible objects such as a table or a banana. A quick description of the classification of DIVE objects, as viewed in Figure 4, follows.

A *view* is what intuitively constitutes an object: Something that has a visual representation in the virtual world. A *light* is not visible in itself, but defines a light source according to different light models, and is applied to the visible views in a world. An *actor* represents a process in a virtual world, such as a user or an application program, and serves as an abstract “reference point” to which messages can be sent through the virtual world. Finally, a *divenode* is a container that may collect zero or more instances of the other object types, along with geometrical transformations for scaling, translating and rotation. A special type of divenode is the *world* object type, which defines the top of a world hierarchy.

For a general introduction to this way of representing objects, see for instance [Foley90, pp 201-226, 285-346]. The approach taken in DIVE is described in [Hagsand95:2].

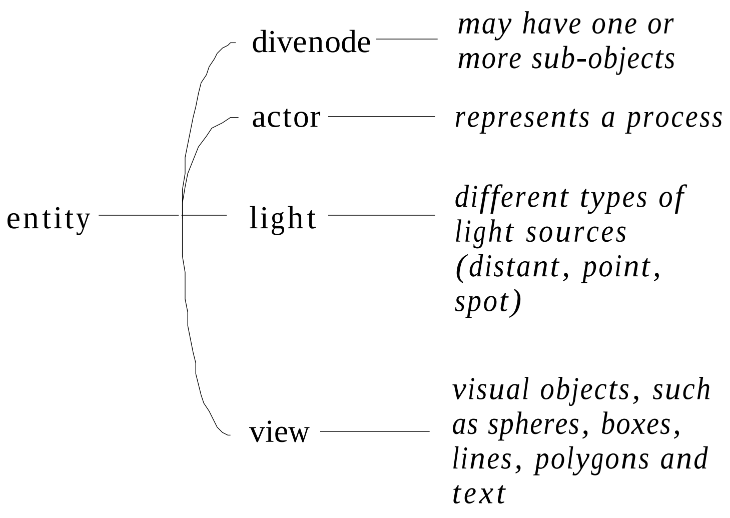
entity
divenode
may have one or more sub-objects
actor
represents a process
light
different types of light sources (distant, point, spot)
view
visual objects, such as spheres, boxes, lines, polygons and text

Figure 4: The main categories in the DIVE entity class hierarchy. See [Hagsand95:2].

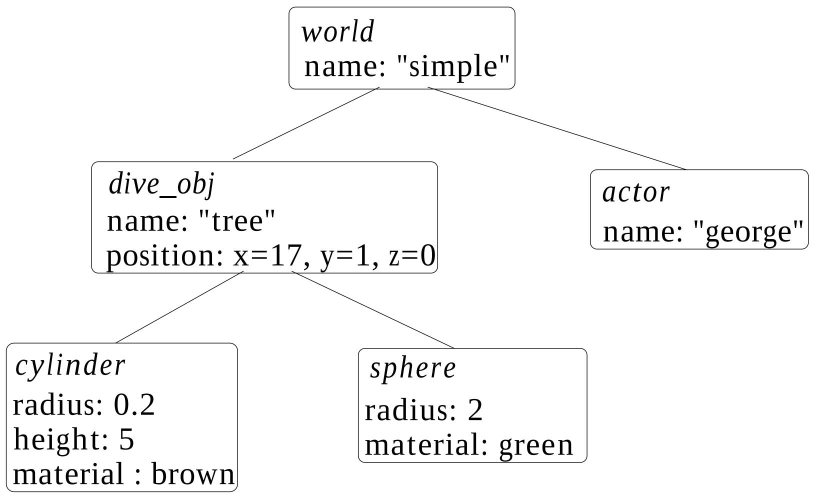

Figure 5: A small world database hierarchy (much simplified).

Complex objects are typically built up of a hierarchy of *divenode* entities, holding together various views that constitute the visual representation. See Figure 5 for an example of a typical, simple world hierarchy. The world object naturally becomes the topmost object in each DIVE world.

Worlds, groups and update messages

A DIVE world is defined by a *multicast group*, to which a process may connect. A multicast group in turn is defined by its network address, and all messages sent to that address will be relayed through the network to each member of the group. This way, a process will only have to send one message to the group instead of repeating the action for each member. (For a discussion on multi-casting, see [Deering89].)

Typically, when a new user process enters a world, it makes a request for the world description from the existing processes in that world (a request for a *state transfer*). Once the world description has been transferred, the local copy of the database is kept up to date with the other processes by exchanging continuous *update messages*, which are sent over the multicast group. These messages reflect events in the virtual world, such as movement or addition of an object.

If a process is the first to enter a world, however, the world description is read from disk. This means that, to ensure that changes to the world will be permanent, the last member of the world should save it to disk before exiting. (This leads to problems when someone wants to enter the world later—where on the network should the world be stored? And how do you know that you are the last process to exit from a world? And what if you don't have enough disk space to store the world? The issue of maintaining persistent virtual worlds using other techniques than a central server is very complex.)

Object behaviours

Objects in DIVE may have *behaviours*, consisting of Tcl scripts directly associated with the objects, and thus residing in the distributed database. Tcl is an interpreted, general-purpose srcipting language, described in [Ousterhout93].

In DIVE, Tcl scripts are simply represented as strings in the object structure and can be triggered when the object is loaded, periodically, or by DIVE events. Such an event may for instance be an *interaction* (typically a mouse click on the object) or a *collision* with another object. Rather complex behaviours may be implemented this way, and access is provided to most of the underlying C functions that form the DIVE platform [Frécon95:2].

Users and processes

Processes may be virtual “applications”, in an everyday sense, using virtual objects as their user interface, and handling the user interaction through the virtual environment. A process can of course be invisible, for instance simply monitoring some aspect of the world and its inhabitants.

An important type of process, *vishnu*, represents a *user*. It provides graphics rendering and sound, and a *virtual body* as a representation for the user in the virtual world—see further the discussion on the interaction model. (The reason for the name *vishnu* is obscure, but it is derived from the older *visualizer*).

The interaction model

Since the key feature of DIVE is its support for collaboration, much emphasis is put on developing metaphors and mechanisms to support this. [Fahlén93]

As mentioned above, each user is represented in the virtual world by a *virtual body*, or *body icon*. Apart from simply representing the user, it also indicates his or her virtual position and viewing angle. In Figure 3 we see some users engaged in a conference. They are represented in the virtual conference room by complex body icons that are capable of displaying different postures depending on what the user is currently doing. A user engaged in interaction with some of the documents on the table will be sitting down on a chair, movement through the conference room will be reflected by “walking” movements of the virtual arms and legs.

DIVE applications

DIVE is implemented in a layered manner, and is presented as a toolkit to use for application programmers. Applications may be programmed and interface with the world in several different manners:

1. As a (UNIX) process programmed in C, using the DIVE C interface, connecting to the world and maintaining its own copy of the database. This is what is usually referred to as a *DIVE process*.
2. As Tcl scripts residing in one or more DIVE objects.
3. As a (UNIX) process, communicating with DIVE by issuing Tcl commands through a UNIX socket. The connection is made to an existing DIVE process (see point 1 above), through the Dive Client Interface (DCI) [Frécon95:1].

The first approach is best suited for “heavier” applications, such as *vishnu* (described above), *MDraw*, and *VR-VIBE* (described below).

The second approach can be used for implementing simple behaviour, such as a rotating object or an object that changes color with time. But since Tcl is a powerful scripting language, complex programs may be implemented. The Tcl script, though, depends on a Tcl shell that must be run by an existing DIVE process. Furthermore, the Tcl scripts in DIVE are interpreted run-time which makes them less suited for making complex computations. This approach has the special property that the scripts are defined in the DIVE objects, and thereby become *elements of the database*. This makes it entirely possible to implement programs that modify the scripts of other objects, inject new scripts, and so on.

The third approach is mainly suited for situations when one does not want a process that is *directly* connected to DIVE, that is to the distributed database itself. Rather, it might be some existing system that connects to a running DIVE process to make simpler queries or exchange information. This approach is suitable for connecting highly complex systems to a DIVE world, without having to redesign them completely.

MDraw and VR-VIBE

As examples of existing applications running in a DIVE world, we may take a virtual whiteboard, and a database browsing and visualisation tool.

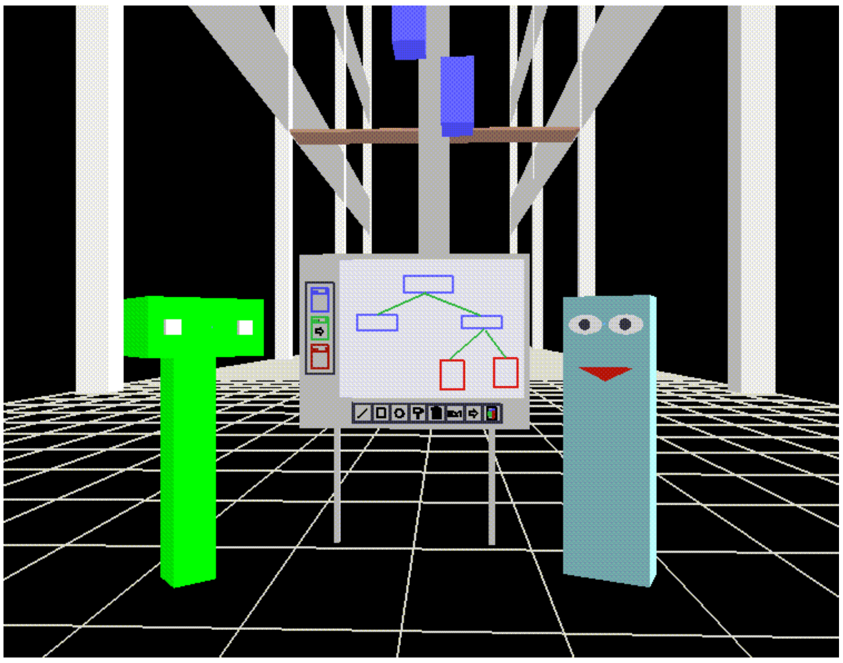

Figure 6: MDraw, a virtual whiteboard along with two users represented by simple polygons. Image from SICS (Swedish Institute of Computer Science), 1992.

MDraw [Ståhl92] is a virtual whiteboard application. It presents itself in the virtual world as a whiteboard, with the addition of a control panel on the left side—comparable to a standard paint program (Figure 6). Any user may approach the whiteboard, select a color from the control panel and draw simple figures on the board. MDraw was one of the very first applications developed using the DIVE system, and, being a collaborative tool, it directly makes use of the most essential feature of system—collaboration.

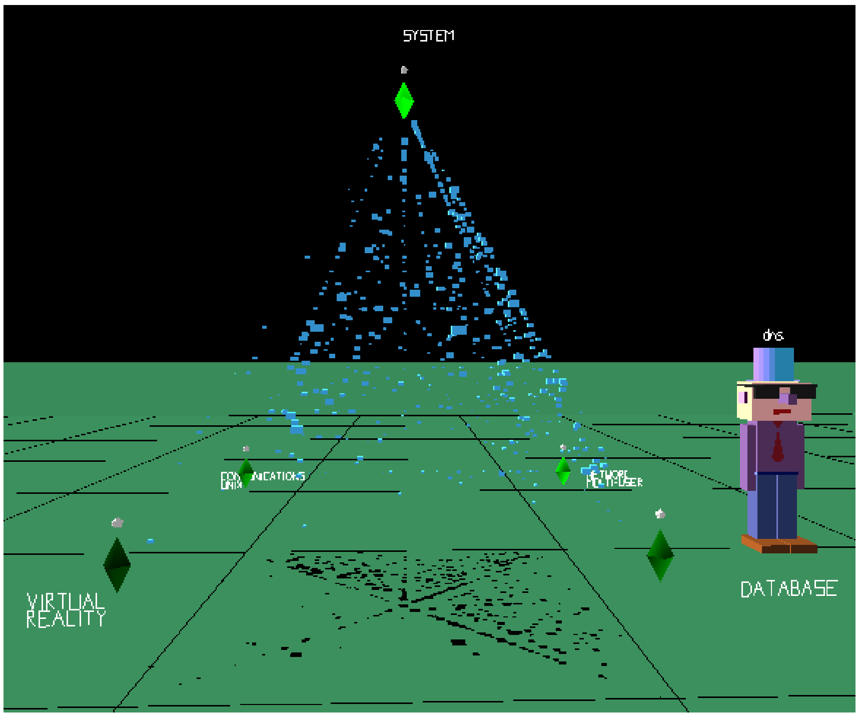

Figure 7: Browsing a bibliography database using VR-VIBE. The larger polygons in the corners of the structure represent the different keywords, and the swarm of smaller objects represent the actual documents—their position reflecting their relative relevance to each of the keywords.

VR-VIBE [Benford95] is an application intended to support browsing and searching of large databases by giving database records properties such as position and size in the virtual world. The idea is that users define several *points of interest* (POI), consisting of a query and a coordinate in virtual space. The database is then searched, and each document representation object is given a 3D position related to how well it matches the query: The better the match, the stronger the "attraction" to the POI in question. The user navigates through the data volume spanned by the POIs, and picks out records for closer inspection by clicking on their representations (Figure 7).

DIVEdit: A prototype

Since the goal of this work is to study the behaviour of users collaborating in a virtual world with the purpose of constructing and modifying objects, we need to provide the possibility of modelling objects directly from within the virtual environment itself. So far, a DIVE user may move objects around, rotate them and (with the support of the *Coloreditor*) change their material.[1] Furthermore, with tools like VR-MOG, DIVE worlds may be constructed in a separate modelling application and, after construction, used in DIVE. Conversion from other file formats, such as RIB (RenderMan) and AutoCad, has made it possible to use many other separate editors to model objects to be used in DIVE. Last, but in no way least, DIVE can directly read VRML files, which is emerging as the standard for specifying 3D objects and scenes on the Internet [VRML96].

This section describes an editor which is to be used *directly* by a user in DIVE, enabling the user to instantly create and manipulate objects from inside the virtual environment (Figure 8). Accordingly, I have called the editor *DIVEdit*—one interpretation of this could be “Distributed Interactive Virtual Editor”, another could be just “DIVE Editor”. The design, implementation and evaluation of this editor is a central part of the work presented in this report.

[1]Since DIVEdit was developed, menu-oriented support for modifying object hierarchies, as well as *textually* editing an object with the vishnu viewer has been implemented. This, however, is done with the standard, two-dimensional Tcl/Tk widget set hardly utilizing the three-dimensional interface other than to select objects and view the changes. But, nevertheless, these functions have turned out to be useful in "everyday" developing and interaction. [DIVE96]

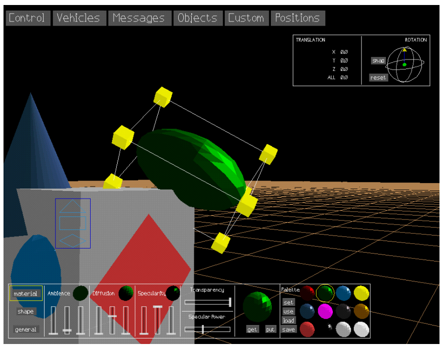

Figure 8: The user interface of DIVEdit. Here the user has selected the round part of the object to change its reflective parameters.

General considerations

Without doubt, there are many interesting approaches to how to take advantage of all the new possibilities given by moving applications into three dimensions. Certainly there is much work to be done here, and much to gain in order to give the user increased power and control when shaping objects in a virtual environment. But at this stage, the purpose of this editor has not primarily been to serve as a test bed for new ways of shaping objects, neither for the development of new metaphors. The two main uses for it so far are:

- A tool for simple modifications of existing worlds and objects. Hereby I mean changing size, shape and material, building objects from existing parts, moving and rotating them etc.
- A multi-user editor to be used for investigating collaborative issues in DIVE, with emphasis on object modelling.

Nevertheless I want DIVEdit to take advantage of the three-dimensional user interface, and, as far as is possible within the scope of this work use ideas from existing work on immersive interfaces, as exemplified in previous sections of this report.

A modest user interface

The users of this first version of DIVEdit supposedly have experience with computers, but perhaps not with virtual worlds. Since they—in an experiment examining multi-user aspects—should not be too concerned with the interface of the editor, but rather feel comfortable with it as soon as possible, the interface should emphasize simplicity before radically new techniques.

For instance, several techniques from existing editors and modellers are well tried out and easily map over from 2D interfaces—such as the use of wireframe bounding boxes and dialogue boxes with buttons and sliders. Even though these may not be optimal solutions for a 3D application, they will serve well for the main purpose of this editor. That is, the interface should be kept *modest*.

But still, while maintaining an interface without too many “surprises” to the user, it is necessary to keep it from becoming too clumsy by overusing buttons, sliders and dials. Whenever possible, an action (such as selecting or rotating a part of an object) should be direct, rather than a matter of clicking around in a dialogue. Developing new such techniques for operations like selecting material, or loading and saving, however, lies beyond the scope of this work.

Means of interaction

The user interacts with DIVEdit in the same manner as is usual with the current DIVE system: By using the mouse to navigate, select and modify the objects. Devices with more degrees of freedom exist and are being developed for use in virtual environments, but DIVE currently supports few of them, being a research tool mainly aimed at examining issues on collaboration and distribution.

Some words on the implementation

Being a DIVE application, DIVEdit has a large and rich infra-structure to rely on. DIVE takes care of complex issues such as distribution of the system, object locking, graphics rendering, handling of input devices and so on. What is presented to an application designer is, simply put, a distributed three-dimensional world, in which users interact with each other and the objects—mainly by selecting and moving different object parts.

Writing an application in DIVE largely becomes a matter of catching events and showing and modifying graphical representations, similar to the way 2D GUI applications are generally written. Some additional attention needs to be given to the question of locking objects, and determining which user has initiated an event. Also, the possibility that any object may be changed or removed by anyone, anytime, is an uncertainty that one needs to deal with.

DIVEdit is implemented as a separate DIVE process that attaches user interface objects to the default setup presented by the vishnu process for each user. The actual communication between different DIVEdit processes is based entirely on the DIVE object database, and is kept as loose as possible to retain the flexibility in moving around, selecting objects, and joining and leaving sessions. Basically, the only actual communication taking place between two DIVEdit processes consists of simple “interference prevention”, such as keeping users from modifying another user's selection marking.

Selecting and modifying objects

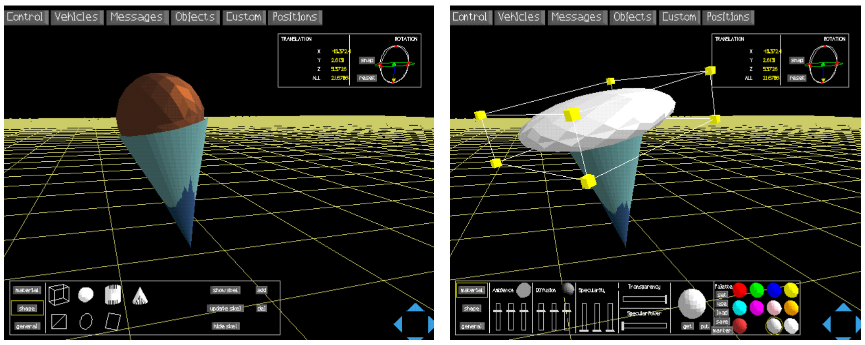

Figure 9: When a part of an object is selected for modification, it is highlighted by a boudning box with handles. To the right, a user has selected and changed the shape of a part of an object.

DIVEdit allows a user to move around freely in the virtual world, and to select any object in it for modification. The selection is made by a simple interaction (a mouse click) with the object of interest. The hereby *selected object* is highlighted by a *selection marking*—a bounding box with eight *handles*. The selected object may be moved and rotated, and by moving the handles also resized (Figure 9). This technique is similar to what has previously been used in for example 3DM (for a review, see the first section of this report).

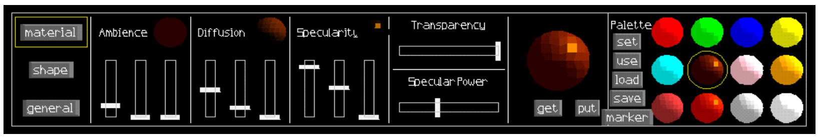

Figure 10: The material modification panel

Using the material modification panel (Figure 10), the surface appearance of the selected object can be modified, with respect to the usual parameters used in 3D computer graphics modelling: Ambient reflection, diffuse reflection, specularity, and transparence. (For a discussion on these parameters, see [Foley90, pp 722-734, 754-758]).

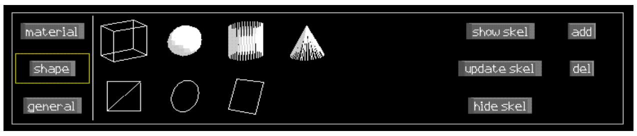

Figure 11: The Shape selection panel

New objects can be created using the shape selection panel (Figure 11). Seven of the basic shapes in DIVE are represented: Boxes, spheres, cylinders, cones, lines, ellipses, and cylinders. In addition, one may want to support general polygons and multipolygons, but this may require more complex interaction for specification and thereby lies a bit ouside the scope of this work.

Spatial and logical structure of objects

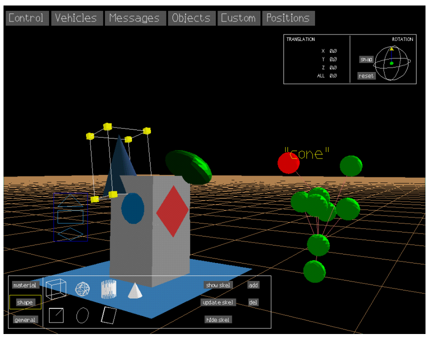

Figure 12: The tree hierarchy to the right visualises the logical structure of the object being modelled. The selected object part is surrounded by the bounding box in the "real" virtual object, and by displaying the object name and changing the colour of the appropriate part of the hierarchy tree.

Since the logical structure of the object (the hierarchy) greatly affects how the object reacts to rotation and movement, it is necessary to present this information to the user. This abstract property can be visualized with a "tree" with nodes and lines, and presented together with the object in the virtual world (Figure 12). The tree can be used for selection, in the same way as for "real" virtual objects, by a simple click on the desired object part. The currently selected part of the hierarchy is highlighted and its name is displayed. In the future this mechanism could be extended to function as a tool for changing the logical structure of the object as well.

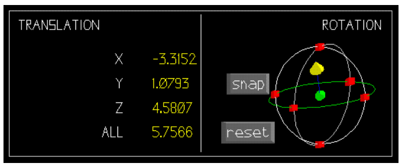

Figure 13: The object information panel is intended to give more detailed feedback on the rotation and position of the selected object. The outlined sphere also functions as a "compass" in that it always is oriented as the world coordinate system, regardless of how the user is viewing the world.

The object information panel is a sketch of a tool to be used for finer positioning control (Figure 13). It also deals with the problem of knowing what is "up" and "down" in the virtual environment, where the user may be rotated at any angle—by presenting a "compass" that is always aligned with the world coordinate system. This tool is very experimental and has not really been evaluated any further within this work.

The problem of scale

DIVEdit works best when the user moves around and modifies some aspects of the environment, adds an object here and there, changes material and alters the size of some part of an object. But modifying large features such as the landscape, roads or house models makes it necessary for the user to move far away from the model, due to the use of bounding boxes. DIVEdit doesn't have any functionality that for instance allows a temporary scaling down of the selected object—the user has to run around large distances to view large objects. An interesting approach to how to address this issue is described in [Stoakley95]: The user holds a miniature 3D “map” of the world in her left virtual “hand”, and may make selections and navigate larger distances by simply pointing at locations in that map.

Further development

There are many reasons to develop DIVEdit further, and many ways to do so. This sub-section lists some of the more significant needs and points of interest, and is of course related to the problems described above.

Use of the aura

Studies are needed to see if there could be a benefit in using the *aura* [Fahlén93] to enhance the user interface of DIVEdit. In short, an aura defines a volume around a virtual representation of a user. For instance, depending on whether the auras of two users intersect or not, different operations and controls could become visible or shared between the users.

Basic functionality

In its present state, DIVEdit only supports a few simple modelling features (see above). If one wants to make it a real tool for modifying DIVE worlds, many features needs to be added. Some functions that need to be implemented are:

- Support for all object types: Text, complex polygons, gateways, grids, light source, …
- Support for more “abstract” object features: Name, visibility flag, wireframe flag, texture, properties, …
- Support for general world features: name, bounds, background colour, …
- Ability to change the hierachical grouping of objects. Ability to freely join two objects, and to split an object.

- Cut, copy and paste of object parts.
- The possibility for a user to select more than just one subobject at a time.

User interface

The user interface of DIVEdit, as it is now, is in many ways incomplete and unsatisfactory. Some things to consider are:

User rotation: The user tests described in the next chapter, as well as everyday experience of designing objects—virtually or in real reality—suggests that a very frequent operation when designing an object is to rotate it to see it from different perspectives. However, in a multi-user environment other users might be irritated if one user keeps rotating an object back and forth—so a different approach is needed.

One could be to move the *user* instead, as on a sphere around the object of interest, always facing the object directly. This way, every user may freely change his viewing angle without affecting other users. However, since the only coordinates that get changed are those of the user, it *could* appear like rotating the whole world. Whether or not this leads to confusion needs to be studied.

Hierarchical structure of objects: As mentioned earlier, it is often difficult to know the internal hierarchical structure of an object just by looking at its exterior. Instead of the current “skeleton view” that is implemented in DIVEdit, one could imagine “dimming” the object by making it wireframed or semi-transparent and then show the skeleton directly inside the dimmed object. This would probably give an even better feeling of how the logical structure of the object relates to its spatial form.

Undo/Redo: Almost any single-user editor of any kind has some mechanism for undoing one or more of a user's operations. In a multi-user application, this becomes much more complex, but must nevertheless be handled in a way that is consistent with existing undo models [Choudhary95]. One must account for situations like when someone wants to undo an operation that another user's operations are depending on. For a discussion on how one might implement undo/redo functionality in a collaborative two-dimensional drawing application (*CoDraw*), see [Avatare95]. The scheme is described in [Prakash92], and is based on keeping history lists with operations, and registering conflicts between operations, that is, operations that cannot be undone due to some later modification.

To conclude, many functions need to be redesigned, and carefully thought over. In my opinion, one should redesign the interface from scratch, and merely see this version of DIVEdit as a first trial.

Practical experiences

The original purpose of this work was to make some "full-fledged" experiments with the object modeller in DIVE. However, the development of the modeller (which was done from scratch) took much more time than expected—a well-known, but seemingly inescapable phenomenon in program development. As a result of that, so far only introductory tests have been performed.

Here, I describe some of the issues that may be investigated in experiments, along with a description of some observations the informal experiment that have been performed, and my own continuous use of the modeller.

Issues of interest

There are numerous questions to be asked about user interaction in virtual environments. Adding the collaborative aspect increases the number by several orders of magnitude. Here are a few issues that I feel are of great interest, and should be subject to further study.

Benefits from collaborative VR: What benefits, if any, are there to draw from the possibility of designing objects in a collaborative virtual environment? What do we gain in comparison to a single-user environment?

Awareness of other users and their actions. How well is the feeling of presence of other users conveyed through DIVE? This is a fundamental question, since the very purpose of DIVE is to serve as a collaborative virtual environment.

Range of visibility: When modifying objects and worlds directly in DIVE, you are always (potentially) in a multi-user environment. What operations should be visible only to you, and what operations should be visible to all of the users in the world? For instance, should everyone be able to see your selection marking? Should only you be able to see your controls, for instance your colour palette? This relates to the ongoing discussion about private objects [Snowdon95], and the confusion that might arise when different users have different views of the virtual world. This issue is addressed further in the last chapter of this report.

Feedback. When should different types of feedback be given? Some operations that give (mainly visual) feedback: choosing an object for editing, selecting parts of the object for modification, changing various attributes to the selected parts, adding and removing objects and object parts. But how do you reflect the changing of the hierarchy of an object?

An informal experiment[1]

This is a description of a simple experiment made with two students who were to sketch a world for showing stellar data using DIVEdit, including getting to know how to use the editor. The purpose was to see how well DIVEdit and DIVE performed in a quite natural situation, where two users were to collaboratively design a world of common interest.

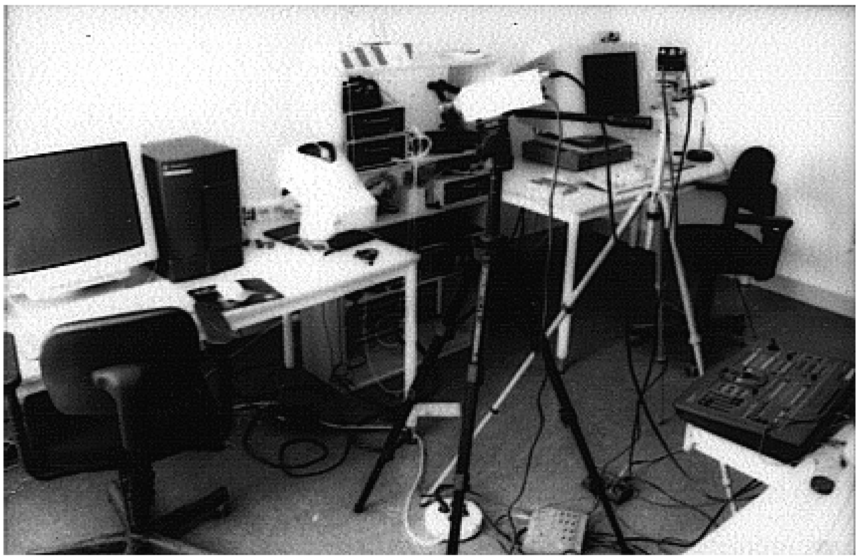

Figure 14: The hardware setup: two SGI workstations, with one camera on each for recording.

[1]This text has, in slightly different form, been published as part of [Stenius95].

Setup

The setup was two workstations, running DIVE and DIVEdit; the workstations located side-by-side, separated by about three meters, see Figure 14. Using this setup, there was of course no need of computer-supported voice communication. The 30-minute session was recorded on video for further reference. Figure 14 depicts the virtual world during the experiment.

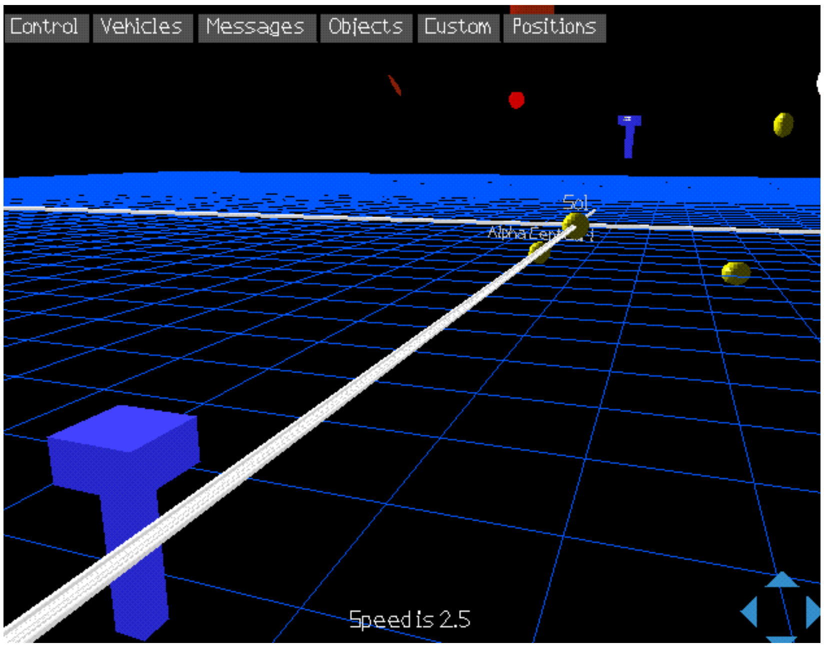

Figure 15: Two users engaged in sketching some objects. (The user representations used here are very simple "T"-shaped avatars).

The task

The task for the users was to get to know the modeller by sketching some objects of their own choice, and then to proceed to sketching on some ideas for a virtual world to be used for visualising stellar data in a project of their own (Figure 15).

Observations

In general, the subjects seemed to benefit from being aware of each others presence in the virtual world, to be able to dicuss the objects freely, and to quickly compare ideas when sketching in the same virtual world. The possibility of being able to locate the other user in the virtual world just "by a glance" seemed to be a great aid in the discussions.

Despite the fact that the subjects were physically in the same room, quite close to each other, they rarely (only a handful of times during an intensive 30-minute session) fell back into looking at each other's screens—and this was mostly due to some unexpected “feature” (or bug) in the software.

When in doubt of the relative alignment of two objects that were modelled, the subjects quickly and without effort solved the problem by standing at perpendicular viewing angles relative to the object, and cooperatively deciding when they were correctly aligned. This is comparable to the way two persons might interact in the real world whe for instance deciding a good position for a painting on a wall. (As mentioned earlier, DIVEdit has no support for “object snapping” or multiple views).

Conclusions from the session

DIVEdit works fine for basic sketching tasks, but fails when more exact control is needed over the placement, size and shape of objects. The general approach of the user interface, though, seemed to work—in particular the now classical direct manipulation approach (selecting objects by clicking, resizing by pulling handles and so on).

Furthermore, and more importantly, the interaction between the users worked well. It seems that the collaborative features of DIVE are of benefit not only on the basic level of presenting users in a virtual world and letting them be aware of each other, but they also allow extension to more complex tasks without becoming confusing.

Discussion

Here, I elaborate on some of the points and results in the work presented in the previous sections. I try to see if there are any conclusions to draw, and give some suggestions for the future. I also discuss some issues that have come up during my continuous use of the modeller regarding user interfaces in VR in general. Most of the issues have some relation to collaborative work, while some touch purely technical issues that I nevertheless feel belong to this report since they came up while I was doing the main work.

Subjective views

In DIVEdit, the selected sub-object is highlighted, as seen for instance in Figure 9, by a wireframe box with graspable corners. Now, if several users are editing the same object, several parts of the object will obviously be highlighted at once. This leads to the need of some way to distinguish between your selection and the other's.

Two ways to achieve this are:

1. By assigning different colours and/or shapes to the selection markings of each user.
2. By providing a way to highlight the private marking, relative to the markings of the other users.

With the first method, you run into the administrative problem of distributing which colors/shapes have been used between different editor processes. And, more severely, you end up with a messy visual appearance, where it may be difficult to distinguish between all the colors and shapes as the number of users increase. And the user gets a potential source of confusion by having to distinguish his or her own colour from the others. This is basically the functionality in DIVEdit today.

The second method, which I argue is the preferred one, could be done by showing each user's own marking corners in a bright, easily detected colour, and the markings of the other users in dull gray, from each user's own view. This would require some way of having *subjective views* of the virtual environment; meaning that the appearance of the environment is different for different users.[1] One big advantage of this method is that the “visual focus” is clearly concentrated to the part of the object that the user is interested in, and supposedly currently editing. For a discussion on subjective views in virtual environments, see for instance [Snowdon95].

Ambiguous object hierarchies

When building a hierarchically composed object, as in DIVE with DIVEdit, I found that it is easy to become confused about how the different parts are related to each other with respect to the underlying hierarchy. When an object is rotated or moved in DIVE, all its sub objects follow it, retaining their internal spatial relation. This direct coupling between basic movement constraints and the underlying structure of the object database is not always natural. This has to do with the simple fact that the idea of hierarchical objects far from always reflects the way we view everyday objects in real life. A human body may be easy to divide into a “hierarchy”, starting with the body, and hierarchically adding head, arms, legs, fingers and so on—simply because of its “tree-like” topology. But what is the top object of a chair? Or a house? Or an amoeba? Once an object have been built, it is very easy to forget its actual hierarchical composition—and become bewildered when editing it later.

Different uses of an object may imply different hierarchies—different objects may be regarded as the “top” object. For instance, if you view a house as part of a *larger group of houses*, the top object may naturally be the actual house itself, an abstract object collecting all the graphical representations that make up the house, the hierarchy following a *conceptual* partition of the house into floors, departments, rooms and so on. But if you view a house as part of its *environment*, the closest superior object being the hill the house is

[1]Since the development of DIVEdit, a new concept called *light-weight groups* has been added to DIVE, among other things enabling the implementation of subjective views of the virtual environment. In short, a light-weight group is a (SID/multicast) group that can be "hooked" onto any object hierarchy in a world, making the hierarchy visible only to the members of the light-weight group.

built upon, a natural top object of the house could be the basement, and the hierarchical composition would more or less following the *physical* way a house is built, from the ground and up.

Refining the user interface

The introductory tests performed with DIVEdit are promising, and show that there definitely are many interesting multi-user issues worth to investigate further. However, it is also clear that in order to as far as possible eliminate disturbing factors, it is absolutely necessary to develop the user interface further. This applies especially to the control of sizes, position and rotation of objects where the simple interaction techniques suitable for navigation and simple interaction have been found to be inadequate in this context.

But the field of user interfaces in 3D is a very new one, and most 3D user interfaces of today are experimental in one way or another. To make my point clear: Compare this with how well developed user interfaces we have for two dimensional displays today—and it becomes clear that there is a need to go deep into developing a basic knowledge of how to design a good interface for an application running on a “three-dimensional display” (that is, in a virtual environment).

Thus, it could be beneficial to first develop a full-fledged modeller—from the very beginning with an interface intended for three dimensions. In this, DIVEdit would naturally serve as a first prototype. When a thoroughly user-tested and working user interface has been developed, it becomes feasible to make extended multi-user experiments and to concentrate on issues related to collaboration.

Using new features in DIVE 3

Since DIVEdit was implemented and tested, DIVE 3 has been released, with a host of new features that provide greater flexibility in application development and user interface building.

The most noteworthy feature is the use of Tcl in DIVE—both with the Tk toolkit for creating dialogs and menus in a desktop environment, and, more notably, the ability to attach Tcl scripts directly to DIVE entities. Both of these features makes it easier to implement DIVEdit in a much more flexible, modular fashion than in its current form:

Interaction methods

As seen in Figure 14, IPLab, where this work was carried out, has access to a range of interaction devices suited for virtual environments, such as the Ascension Bird, the Immersion Probe, and a head-mounted display with head tracking. Using devices such as these in shaping objects in virtual environments opens up many possibilities and gives rise to many interesting questions regarding user interfaces, ergonomics, and so on—and is thereby a possible area for further development of DIVEdit, which for now does not take direct advantage of any of these interaction methods. Some of the related work mentioned in the section on World and object modellers serve as good examples of this area of research.

A look into the future

To round off, there is a host of issues related to collaboration and user interfaces that should be subject for further examination. Some interesting questions that have shown to be relevant are:

- How do you distinguish between the result of your own actions and others? With many users actively modifying a virtual world, the modification events may become numerous and rapid—with a risk of "losing track" of the result of your own commands among many other modifications.
- How do you avoid confusing interference when several users are modifying the same object?
- To what extent should feedback be distributed to all users? How does this depend on the current context and action?

Collaborative sketching

An interesting example, that has occured after the actual development and testing of DIVEdit, is AlphaWorld [Alpha96]. Developed at Worlds, Inc. AlphaWorld is a collaborative VR system, server-based and with a graphic rendering tuned for regular PC hardware. Of specific interest in AlphaWorld is the collaborative building of virtual towns and surroundings that has taken place, having the characteristic of social gatherings where a common idea or concept is made (virtually) real—such as a "recreational" area around a lake where people can build their own virtual "summer houses". AlphaWorld is interesting in many aspects, especially social, and draws ideas from Multi-User Dungeons (MUD:s)—text-oriented multi-user games over Internet—and applies similar rules and schemes to a three-dimensional environment.

Regarding DIVEdit, it has been said[1] that the most important use for such an immersive modeller is fast, collaborative *sketching* of

[1]In informal discussions with people mentioned in the beginning of this report.

objects and environments—and that this is the general way we should move when developing the idea further. Both spontaneous comments from people confronted with DIVEdit, and the simple user tests I have described above support this view.

References

(Note that some of these refer to documents located on the World Wide Web (WWW), and thus their positions are likely to change. Even though the information hopefully still is available, consulting your favourite search engine may be required).

[Alpha96] "AlphaWorld(TM)", *WWW:* http://www.worlds.net/alphaworld/, Worlds Inc. 1996

[Avatare95] Anneli Avatare, "Multi-user drawing editors", Licentiate thesis in computing science, Department of Numerical Analysis and Computing Science, Royal Institute of Technology and University of Stockholm, Sweden, 1995, ref TRITA-NA-9504

[Benford95] Steve Benford, Dave Snowdon, Chris Greenhalgh, Rob Ingram, Ian Knox and Chris Brown, "VR-VIBE: A virtual environment for co-operative information retrieval", *Eurographics'95*, Maastricht, The Netherlands, August 30 – September 1 1995

[Bier90] Eric A. Bier, "Snap-dragging in three dimensions", *Proc. Workshop on Interactive 3D Graphics*, Showbird, Utah, 1990, pp 193–204

[Butterworth92]Jeff Butterworth, Andrew Davidson, Stephen Hench, and Marc Olano, "3DM: A three dimensional modeler using a head-mounted display", *Proc. 1992 Symposium on Interactive 3D Graphics*, Cambridge, Massachusetts, March 29 – April 1 1992, pp 135–138, 226

[Buxton86] William Buxton, Brad A. Myers, "A study in two-handed input", *CHI'86 Proceedings*, April 1986, pp 321–326

[Carlbom78] Ingrid Carlbom, Joseph Paciorek, "Planar geometric projections and viewing transformations", *Computing Surveys*, 10, December 1978, pp 465–502

[Carlsson92] Christer Carlsson and Olof Hagsand, "The MultiG Distributed Interactive Virtual Environment", *Proc. 5th MultiG Workshop*, Royal Institute of Technology, Stockholm, Sweden, December 1992

[Carlsson93] Christer Carlsson and Olof Hagsand, "DIVE—a multi-user virtual reality system", VRAIS'93, IEEE Virtual Reality Annual International Symposium, pp 394–400

[Choudhary95] Rajiv Choudhary, Prasun Dewan, “A general multi-user undo/redo model”, *Proc. 4th European Conference on Computer-Supported Cooperative Work*, Stockholm, Sweden, September 10–14 1995, pp 231–246

[Colebourne96] "VR Mog” application developed by Andy Colebourne, Lancaster Universtity, *WWW:* http://www.comp.lancs.ac.uk/computing/research/cseg/projects/vrmog

[Deering89] S. Deering, “Host extensions for IP multicasting”, STD 5, RFC 1112, Stanford University, August 1989

[DIVE96] "The DIVE Home Page”, *WWW:* http://www.sics.se/dive/, Swedish Institute of Computer Science, Stockholm, Sweden, 1996

[Fahlén93] Lennart E. Fahlén, Charles Grant Brown, Olov Ståhl, Christer Carlsson, “A space based model for user interaction in shared synthetic environments”, *INTERCHI'93*, April 24–29 1993, pp 43–48

[Foley90] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, *Computer graphics: principles and practice*, 2nd ed, Addison-Wesley, 1990

[Frécon95:1] Emmanuel Frécon and Olof Hagsand, “The Dive Client Interface”, Reference Document, *WWW:* http://www.sics.se/dive/manual/dci.html, Swedish Institute of Computer Science, November, 1995

[Frécon95:2] Emmanuel Frécon and Olof Hagsand, “The Dive/Tcl Behaviour Interface”, Reference Document, *WWW:* http://www.sics.se/dive/manual/tcl-behaviour.html, Swedish Institute of Computer Science, November, 1995

[Gisi94] Mark A. Gisi and Cristiano Sacchi, “Co-CAD: A collaborative mechanical CAD system”, *PRESENCE*, 3, Fall 1994, pp 341–350

[Hagsand95:1] Olof Hagsand, “SID2 interface specification”, Reference Document, *WWW:* http://www.sics.se/~olof/sid2.html, Swedish Institute of Computer Science, August, 1995

[Hagsand95:2] Olof Hagsand, “Dive entity interface”, Dive 3 Reference Manual, *WWW:* http://www.sics.se/dive/manual/entity.html, Swedish Institute of Computer Science, September, 1995

[Hagsand96] Olof Hagsand, “Interactive Multiuser VEs in the DIVE system”, *IEEE Multimedia*, Spring 1996, pp 30–39

[Houde92] Stephanie Houde, “Iterative design of an interface for easy 3-D direct manipulation”, *CHI'92*, May 3–9 1992, pp 135–141

[Liang93] Jiandong Liang and Mark Green, “Geometric modeling using six degrees of freedom input devices”, *Proc. 3rd International Conference on CAD and Computer Graphics*, Beijing, China, August 23–26 1993, pp 217–222

[Ousterhout93] John K. Ousterhout, *TCL and the TK Toolkit*, Addison-Wesley, 1993

[Prakash92] Atul Prakash and Michael J. Knister, “Undoing actions in collaborative work: Framework and experience”, *Proc. CSCW'92*, Toronto, Canada, October 31 – November 4 1992, pp 273–280

[Sachs91] Emmanuel Sachs, Andrew Roberts, and David Stoops, “3-Draw: A tool for designing 3D shapes”, *IEEE Computer Graphics & Applications*, 11, November 1991, pp 18–26

[Shu94] Li Shu and Woodie Flowers, “Teledesign: Groupware user experiments in three-dimensional computer-aided design”, *Collaborative Computing*, 1, 1994, pp 1–14

[Snowdon95] Dave Snowdon, Chris Greenhalgh and Steve Benford “What you see is not what I see: Subjectivity in virtual environments”, *Framework for Immersive Virtual Environments (FIVE'95)*, London, UK, December 18–19 1995

[Stenius95] Mårten Stenius, “An object modeler for distributed virtual worlds”, poster / short paper, *4th European Conference on Computer Supported Cooperative Work 1995, Conference Supplement*, Stockholm, Sweden, September 1995

[Stoakley95] Richard Stoakley, Matthew J. Conway, Randy Pausch, “Virtual Reality on a WIM: Interactive Worlds in Miniature”, *Proc. SIGCHI'95*, *WWW:* http://www.cs.virginia.edu/~rws2v/wim/wim-paper.html

[Ståhl92] Olov Ståhl, “Implementation issues of aura based tools”, *Proc. 5th MultiG Workshop*, Royal Institute of Technology, Stockholm, Sweden, December 1992

[VRML96] Gavin Bell, Anthony Parisi, Mark Pesce, “The Virtual Reality Modeling Language—Version 1.0C Specification”, *WWW:* http://vag.vrml.org/vrml10c.html