Defining Interaction within Immersive Virtual Environments

by

Anthony James Steed

Submitted to the University of London for the degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

Queen Mary & Westfield College

University of London

Mile End Road

London E1 4NS

September 1996

Abstract

This thesis is concerned with the design of Virtual Environments (VEs) - in particular with the tools and techniques used to describe interesting and useful environments. This concern is not only with respect to the appearance of objects in the VE but also with their behaviours and their reactions to actions of the participants. The main research hypothesis is that there are several advantages to constructing these interactions and behaviours whilst remaining immersed within the VE which they describe. These advantages include the fact that editing is done interactively with immediate effect and without having to resort to the usual edit-compile-test cycle. This means that the participant doesn't have to leave the VE and lose their sense of presence within it, and editing tasks can take advantage of the enhanced spatial cognition and naturalistic interaction metaphors a VE provides.

To this end a data flow dialogue architecture with an immersive virtual environment presentation system was designed and built. The data flow consists of streams of data that originate at sensors that register the body state of the participant, flowing through filters that modify the streams and affect the VE.

The requirements for such a system and the filters it should contain are derived from two pieces of work on interaction metaphors, one based on a desktop system using a novel input device and the second a navigation technique for an immersive system. The analysis of these metaphors highlighted particular tasks that such a virtual environment dialogue architecture (VEDA) system might be used to solve, and illustrate the scope of interactions that should be accommodated.

Initial evaluation of the VEDA system is provided by moderately sized demonstration environments and tools constructed by the author. Further evaluation is provided by an in-depth study where three novice VE designers were invited to construct VEs with the VEDA system. This highlighted the flexibility that the VEDA approach provides and the utility of the immersive presentation over traditional techniques in that it allows the participant to use more natural and expressive techniques in the construction process. In other words the evaluation shows how the immersive facilities of VEs can be exploited in the process of constructing further VEs.

Acknowledgements

Many people have influenced the final form this thesis has taken, and the author would like to take this opportunity to thank them.

I have been fortunate to have worked in such an interesting field over the past few years with an excellent supervisor in Mel Slater. His knowledge and attitude towards the subject have made it a continuously stimulating and frequently entertaining area for study.

I would like to thank all my contemporaries in the ACE lab at QMW whose diverse interests and backgrounds made working in that environment so stimulating. Particular thanks go to Yiorgos, Mark, Martin, Alastair and Amela, for service beyond the call of duty in their criticisms, technical assistance and proof-reading at various stages of my Ph.D. Thanks also to all those at QMW who made it a friendly place to work and to everyone who participated in the reported studies.

At a personal level I would like to thank anyone I have ever bagged a mountain with. Elevated altitude is an excellent cure for stress.

Thanks also to Rachel for her love and understanding, and for not being at all scary really.

Finally, thanks must go to my parents for their love and support over the course of my study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Looking at the history of computing we can see a transition from systems based on complex command languages to what Shneiderman calls 'direct manipulation' systems [Shn83]. According to Shneiderman the defining features of such a system are the real-time display of the object of interest, rapid, incremental and reversible actions, and direct manipulation of the object of interest. Thus the object of interest is displayed and the user can interact with it using physical gestures, unmediated by a command line interface. Minsky sums up the aim of such a system as [Min84]:

> ...to create worlds within the computer that can be manipulated in a concrete way using gesture as the mode of interaction. The effect is intended to have a quality of "telepresence" in the sense that, to the user, the distinction between real and simulated physical objects displayed on a screen can be blurred ...

Minsky is describing the Fingerpaint system, a force sensitive display panel which could sense planar position of the point of finger contact on the screen and a three dimensional force vector. This extension of a touch panel display allowed many painting effects to accomplished with a very simple interface.

This sense of blurring the distinction between real and simulated physical objects is one of the ideas driving the construction of three dimensional interactive interfaces, especially those using the virtual reality paradigm. The belief is that with a three dimensional representation people will be able to use or adapt previously learned skills, so that the interface will be more transparent and natural. This doesn't necessarily rule out fantastic environments but, as we shall see in Chapter 3, the interaction for an environment should include a representation of the user and the interaction metaphors

employed within the environment and the behaviours of the objects should be consistent and comprehensible by the user.

1.1 Virtual Environments

The core of the system is a virtual environment (VE) database, and we shall be particularly concerned with virtual environments that have three-dimensionality. The VE database must model three things [Ell91]:

- *Content* This the objects and actors of the environment. Objects have properties, such as position and appearance. Actors are objects that have the capacity to initiate interactions with other objects.
- *Geometry* This is a description of the space in which the content is positioned. It has dimensionality, metrics and extents. The dimensionality refers to the number of terms needed to specify a position. The metrics are rules that give an ordering of the positions to establish geodesic or straight lines in the space. The extent is the range of values which the positions might take.
- *Dynamics* Are the rules governing the interaction between the contents of the environment. Examples include simulations of physical laws or gesture interpretation.

The VE database is maintained and updated using the dynamical rules of the environment and the effects of the interaction of one or more users with that environment. A user is experiencing this VE in one or more sensory modalities, through display devices that are showing a rendered view of the environment. This rendering depicts the environment and creates a sensory impression of it with respect to some model of the participant within that environment. This model of *self* is derived from sensors on the participant's body that report some aspects of the state of the participant, for example the position of limbs or a gesture mediated by a desktop input device such as a mouse.

Together these components form a virtual environment generator, the structure of which is shown in Figure 1.1.

1.2 Interaction with Virtual Environments

The participant's actions are mediated through a body model into actions within the virtual environment. At a basic level this model provides two

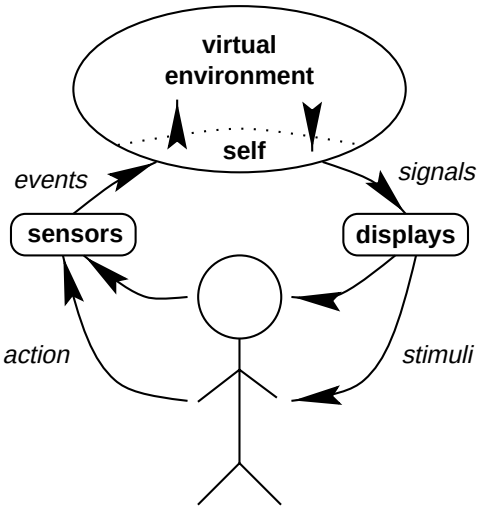


Figure 1.1: Components of a virtual environment generator

elements a *viewpoint* and a *cursor*. The viewpoint is the point at which the participant is considered to be within the VE and provides a locale from which the rendering of sensory information can take place. The cursor is a simple extension of the two dimensional cursor to three dimensions. It is an effector within the VE and it used to interact with the content.

Interaction with a three dimensional environment then involves a combination of the following tasks [SD91]:

1. Navigation - the user should be able to navigate about the VE by controlling the viewpoint.

2. Global selection - the user should be able to select any object in the VE.

3. Rigid body transformation - this includes translation and rotation of an object, and any other transformation which changes the object's position in space, but leaves its other properties unchanged.

4. Local selection - the user should be able to select part of an object which can then be used to deform the object but not its overall position in space.

5. Deformations - these transformations applied to an object cause it to deform either uniformly (for example by scaling) or non-uniformly, by twisting, tapering or bending. This actually affects the properties of the

environment, whereas rigid body transformation just alters the position of an object.

How the user performs these tasks depends on two main criteria, the input devices that sense the user, which range from desktop mice to full body posture sensing suits, and the interaction metaphor that is employed. The interaction metaphor will obviously depend upon the devices used simply because some devices will only sense a small number of parts of the user's body with few degrees of freedom, fewer possibly than the number required to specify navigation or object manipulation without extra interaction modes or limitation in the task.

In this thesis we will distinguish between *immersive virtual environments* (IVEs) and *non-immersive virtual environments* (NIVEs). The difference between these is a question of both interaction style and the representation of self within the environment.

The rendering of an IVE is slaved to the body of the participant, so that the rendered displays match as closely as possible the changes that would be expected in a real environment when the same motions were made. In particular the visual viewpoint is slaved to the head of the participant, so the graphical display is continuously updated. The viewpoint from which a NIVE is rendered does not match the participant's body but is controlled indirectly through input devices. Similarly a cursor or representation of the body of the participant will be slaved to the sensor devices in an IVE system, but controlled indirectly by some appropriate interaction metaphor in an NIVE system. Also, in an IVE system, if both the hand and head are tracked then they will be modelled inside the VE in the correct relative positions so that the proprioceptive sense of where the head and hand are matches the display.

1.3 Scope

This thesis is concerned with the design, implementation and evaluation of an immersive virtual environment, within which it is possible to describe interactions of the participant and behaviours of the component objects of the environment.

The basic requirements for such a system are developed from consideration of interactions with IVE and NIVE systems. These generate technical requirements and also illustrate the scope of interaction techniques that must be covered. The study of a NIVE system demonstrates how the device used has an affect on the interaction techniques used. The IVE study demonstrates how the semantics of the virtual environment dictate interaction metaphors

and how, to some extent, the use of a virtual body makes the interaction device dependent.

From consideration of current programming systems for virtual environments and current work in the field of visual programming, a design of a programming system that integrates the programming and display environments is produced. This programming system, the Virtual Environment Dialogue Architecture (VEDA) system is designed as an immersive application in order to exploit the advantages an IVE gives over a NIVE for three dimensional structure comprehension and task performance.

The prototype implementation demonstrates the advantages of integrating the VE with its behavioural description in a data flow model, over more traditional programming or scripting for virtual environments, particularly in that the participant doesn't have to leave the VE in order to make changes.

1.4 Contributions

The main contribution of this thesis is the design, implementation and evaluation of the VEDA system. This allows the construction and modification of interaction techniques and object behaviours from within the VE. It provides a meta-view of an operating VE in order to allow 'expert' users to manipulate a structure representing the definition of the interactions. Therefore this thesis covers:

- Reviews of the relevant literature on gestural interaction, visual programming and VE systems.
- A case study of interaction with a NIVE system focusing on a input device called the Desktop Bat.
- A case study of interaction with an IVE system using a Virtual Treadmill metaphor.
- Discussion of the effects interaction has on immersion and presence within a VE and VEDA's role within a framework for supporting the sense of presence.
- Design of VEDA with reference to current VE description languages.
- Evaluation of VEDA.

1.5 Structure

This thesis is organized in the following chapters:

Chapter 2 introduces the devices and metaphors used to interact with NIVEs. Using a device called the Desktop Bat as an example it highlights some of the problems inherent in designing natural metaphors for interaction and the limitations of desktop interaction.

Chapter 3 describes metaphors for interaction with immersive virtual environments. It discusses what presence is, and ways of measuring it, and then describes experiments to evaluate some aspects of virtual environment design that affect presence. It then describes in detail the design of the *Virtual Treadmill Metaphor* and it's ease of use and effect on presence. The chapter concludes by discussing some extensions to the virtual treadmill metaphor and a more general model of presence.

Chapter 4 discusses virtual environment description, both in terms of a model of the virtual environment's geometry and object properties and in terms of the specification of object behaviour and the participants interactions with the environment. It also reviews work in visual programming that will be relevant for the design of the programming system that is the main focus of the thesis.

Chapter 5 describes the design of the virtual environment programming system and justifies its being presented immersively. It describes the range of services provided by such a system with reference to current programming systems for virtual environments, and how these are integrated into a data flow model. The implementation of the system is discussed and the design decisions taken when presenting the tools to manipulate the data flow model while within the environment.

Chapter 6 provides a three-fold evaluation of the system. Firstly it evaluates whether the system presented solves the problems that Chapters 2 and 3 raised concerning the design and customization of interaction metaphors. Secondly it presents the design and construction of an application with this system to highlight the capabilities it affords. Thirdly a user case study shows how, for naïve users, this system simplifies the construction of applications from scratch and their modification.

Chapter 7 concludes and indicates how the capabilities of VEDA can be expanded to cover, for example, distributed, multi-user applications.

Chapter 2

Desktop Interaction

This chapter investigates techniques for interacting with a desktop or non-immersive virtual environment (NIVE). The investigation demonstrates how the input device can affect the range of possible interaction metaphors that can be used. The research described in this chapter thus leads to the conclusion that VEDA should not impose an interaction style or policy on the virtual environment (VE) designer.

The chapter reviews material relating to interaction devices and metaphors, and discusses in detail the design and use of a device called the Desktop Bat for interaction with NIVEs [SS95]. An experiment compares several metaphors for both *navigation* and *object manipulation* with the Desktop Bat in three different scenarios and gives some conclusions about the best metaphor for each task. Not only are these basic metaphors different but it is shown that picking and placing of objects are best accomplished with different metaphors, and thus there is no one interaction metaphor that is suitable for all tasks. This leads to a discussion of a general limitation of NIVEs interaction techniques in that they are quite tightly tied to a particular device. Later in Chapter 3 we shall see that for an IVE the interaction techniques are largely device independent.

In this chapter Section 2.1 reviews desktop interaction, the devices that are used and techniques for navigation and object manipulation. Section 2.2 describes the construction of the Desktop Bat and the metaphors for interaction selected for investigation. Section 2.3 describes the design and Section 2.4 the results of an experiment to test these interactions metaphors. Conclusions about the Desktop Bat are given in Section 2.5 and finally Section 2.6 describes some limitations of the Desktop Bat and desk bound interaction in general.

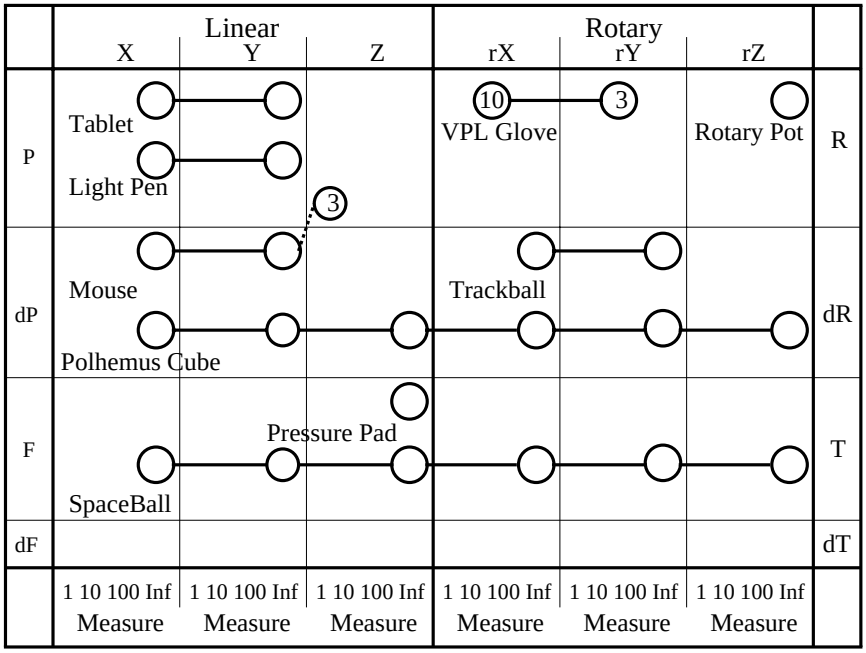


Figure 2.1: Some input devices plotted using the taxonomy of Mackinlay *et al.*

2.1 Non-Immersive Interaction

Many applications now demand interaction with 3D data-sets and the commonest display system is still the desktop workstation. Interaction with desktop display systems is also usually based on mice because of their ubiquitous nature and many methods have been developed that use the mouse to perform navigation and object manipulation within the virtual environments.

The task of specifying an object or viewpoint position has six degrees of freedom, and so any device that senses fewer degrees will have to be used with a metaphor that compensates for the lack of dimensionality. However for NIVEs we shall see that having a 6 degree of freedom device on a desktop system may not be ideal and that devices with fewer degrees of freedom may be more suitable.

A comprehensive review of desktop interaction devices is beyond the scope of this chapter, the reader is referred to the taxonomies of Buxton, Foley and Mackinlay *et al.* [FWC84, BM86, MCR90]. Using the taxonomy of Mackinlay *et al.* and examples from Buxton, Foley and Mackinlay*et al.* a few demonstrative devices have been illustrated in Figure 2.1.

The taxonomy breaks devices down into individual components that they sense, classifies these components and then composes them back into com-

plete description of the device. Each circle on the diagram represents a transducer of a physical property. The classification is

- Linear/Rotary
- Position/Force
- Relative/Absolute
- Direction
- Sensitivity

Thus a simple button is a linear sensor, that measures relative position in the Z direction[1] at a sensitivity of one, that is it senses a discrete value.

Two types of composition are used in this figure [2] *Merge composition* and *Layout Composition*. Merge composition of two devices creates a single device that is considered to generate output in the merged domain of both original devices simultaneously. Layout composition of several devices generates a single device that can sense the separate properties of each device separately. A button panel is so composed since each button is independently sensed and can be operated individually.

In Figure 2.1 merge composition is indicated by a solid line, and layout composition by a dotted line. A tablet or light pen is thus a merge of two absolute position sensing devices, one in the X direction, one in the Y direction. A mouse is similar, but it senses relative position, since it can be picked up and moved without the planar position being registered, and is usually layout composed with a number of buttons.

Several other popular devices are plotted, though there are probably several variations for most of the devices, particularly where there is a possibility of adding buttons.

The Polhemus Fastrak^R [Pol] and Spaceball^R 2003^{TM} [Spa], are radically different devices though each senses the same number of degrees of freedom. The Polhemus Fastrak^R is a magnetic tracking device that consists of a base transmitter that generates a switching magnetic field which induces current in a receiver device from which the relative positions of the receiver and transmitter can be deduced. The Spaceball senses force and torque applied to a ball fixed to a stationary base. The Spaceball's advantage over the

[1] For the purposes of this exposition, a body centred axis system is used, X being to one's right, Y away and Z up.

[2] Mackinlay *et al.*'s taxonomy also describes a third composition, *Connection composition* where the output of one device is piped into the input of a second device which is often a virtual device such as a virtual slider which then generates a further output value.

Polhemus Fastrak or any free space tracking device is that it rests on the desk and doesn't need to be held in the air. The following section describes general technique to alleviate this problem.

Also shown in the diagram is the VPL dataglove [Zim85, ZL87], which has been used in desktop systems though it is more usually associated with IVE systems. It senses the angle of bend for the two proximal joints of each digit, and has an option for sensing the abduction of the thumb and first and middle fingers. Overall this makes it a 13 degree of freedom sensing device.

2.1.1 Object Positioning Techniques

As mentioned, the task of positioning an object has six degrees of freedom with three for the location and three for the orientation.

With two degree of freedom devices being so widespread, many approaches have used mice and joysticks to control objects either directly using mode switches to enable translations and rotations in different directions, or through the use of virtual devices. Nielson and Olsen take the 2D locator movement and map this on to motion along the virtual world coordinate axis which is closest to the direction of movement in the current 2D projection [NO86]. Thus the plane of motion of the 2D locator is separated into six regions which correspond to motion in the positive and negative directions along the coordinate system axes. This allows placement of a 3D cursor and then rotations can be specified in terms of this and other 3D points.

Chen *et al.* give four metaphors for the direct control of the rotation of an object [CMS88]. The most efficient metaphor is that of a virtual sphere where the object is imagined to be contained within a glass sphere. Up, down, left and right motions of the 2D control device roll the sphere in the appropriate direction and rotation about the remaining axis occurs when the user moves the device along a circular path. Specifying 3D points manually is time consuming and difficult to perform accurately so Bier combats this by allowing the cursor to be attached to object vertices, lines or faces. This allows very accurate positioning and most translations and rotations can be described relative to objects that already exist [Bie86, Bie90].

With the development of six degrees of freedom position sensors [MAB92], a more direct metaphor of using hand location and orientation for object location and orientation becomes available. However it is not obvious that such a technique is ideal for long periods of interaction with a NIVE, with their limitations of accuracy and tendency to be tiring to use [BMB86, WJ88, War90]. Badler *et al.* found that using the absolute position of the hand as the cursor position could become tiring for the user [BMB86]. They overcame this by having a button in the user's free hand that would engage the device

in a relative mode. They also found that it was extremely difficult to keep the cursor stationary or to move it in just one dimension. Ware and Jessome also considered the problem of object placement with a polhemus device and they found it to be an easy task to perform roughly [WJ88]. Their addition of a 90^o rotation of the viewing position about the vertical axis in combination with a disablement of movements perpendicular to the viewing plane produced a metaphor that was simple and effective to use. Ware goes on to compare this mode with the all degrees of freedom mode and compare stereo-scopic with mono-scopic displays [War90].

2.1.2 Navigation Techniques

Mackinlay *et al.* identify four main types of navigation within 3D scenes [MCR90].

1. General movement. Exploratory movement around the model
2. Targeted movement. Movement with respect to specified targets, such as moving toward an item of interest.
3. Specified coordinate movement. Movement to a precise position and orientation.
4. Specified trajectory movement. Movement along a position and orientation trajectory.

We are concerned with the first type, unconstrained exploration of an environment and this is six degree of motion task, with three degrees to specify rotation and three degrees to specify position.

Two basic techniques exist, moving the viewpoint through the workspace or moving the workspace around the viewpoint. Essentially the difference is the choice of coordinate system in which to perform translations and rotations. Ware and Osborne give three navigation metaphors that illustrate the difference [WO90].

- Eyeball in hand. The movements of the input device correspond directly to movements of the eye.
- Scene in hand. The scene itself is slaved to the input device.
- Flying vehicle control. The input device provides the controls for the vehicle such as velocity and rotation.

Ware and Osborne note that the scene in hand metaphor is a poor choice for complex environments because the centre of rotation is fixed and this leads to confusing translations during rotation. Choosing the centre of rotation would be possible using the techniques of cursor placement as described previously.

Once again, two degree of freedom input devices can be used for navigation, either directly or through virtual devices [RCM89, WS91]. An example is using a mouse to point out on the display the required direction of movement, and using a different mode to control rotations.

The use of a 6D tracking device allows the view to be slaved to the head position [Sut68, FMHR86, CHB+89, McK92], a technique that McKenna indicates improves the ability to pick locations in space [McK92]. Head tracking systems for desktop environments have been described as *fish tank virtual reality* [WAB93], since the 3D environment is limited by its' being projected by a static screen giving a small working area.

However, because of the limited range of position trackers, the user is often constrained to stay within a small area and thus metaphors have to be used to navigate over longer distances [BBH+90, BHV92].

For movement over long distances velocity control can be used [Fis90, WS91, CW92, SN93]. Velocity control allows rapid movement over large distances, but is inaccurate when approaching an object, though this problem can be overcome by use of a logarithmic approach technique [MCR90]. This technique works when there is a point of interest, for more general velocity control Chapman and Ware's predictor based visual feedback aid can help users learn how to control the viewpoint more effectively [CW92].

2.2 The Desktop Bat

The Desktop Bat consists of a dome attached via three joints to a mouse base [SD91]. The dome rotates in three directions and combined with the planar location of the Desktop Bat this makes it a five degree of freedom device (see Figure 2.2). The Desktop Bat also has five buttons that are placed under the natural resting places of the fingers and thumb.

In the taxonomy of [MCR90] described in Section 2.1, this device's position is illustrated in Figure 2.3. However it is easy to rotate without moving since the base is fairly weighty compared to the dome, but hard to move without rotating since the force to push the Bat is usually applied through the dome's universal joint, though it is possible to position the dome at an extremity of movement which prevents it from rotating while being moved on the plane.

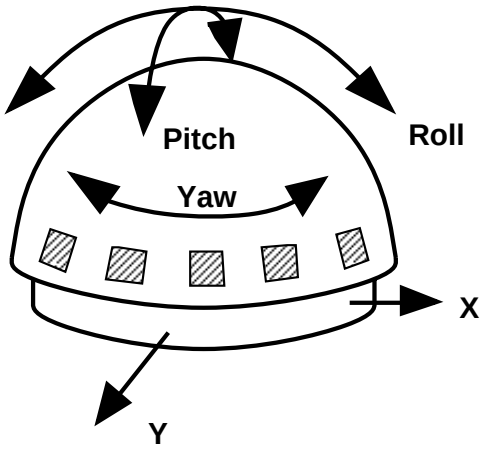


Figure 2.2: Degrees of freedom of the Desktop Bat

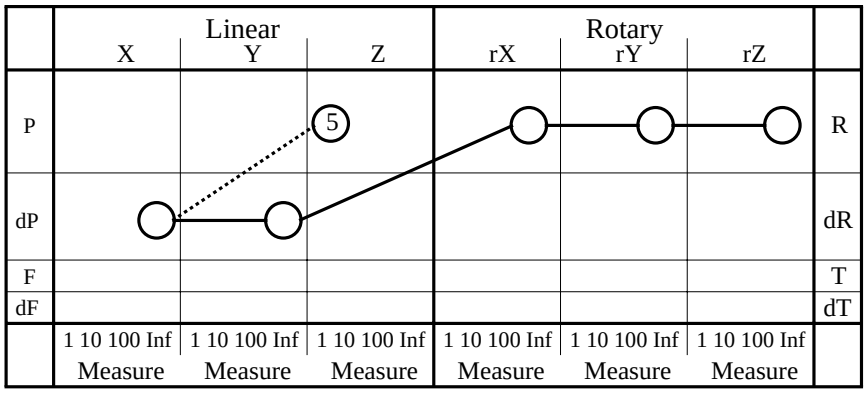


Figure 2.3: The position of the Desktop Bat in the taxonomy of Mackinlay *et al.*

The conventions used in the following descriptions of the interaction metaphors are: the world coordinate system is defined by the axes X,Y and Z, where for exposition we are taking the vertical direction in the scene to be parallel to the Z axis; the eye coordinate system is defined by the axes U,V and N where the line of sight is along the N axis and up relative to the eye is along the V axis.

2.2.1 Navigation

The Bat has five degrees of freedom, but the task of navigation requires 6 degrees of freedom, with 3 required to specify the location and 3 for the orientation of the eye point. Three metaphors that allow general navigation are:

1. **Hand on Eye**. In this metaphor rotations of the Bat cause rotations of U,V and N axes. For example roll of the Bat corresponds to rotation of the U and V axes around the N axis. Planar movement of the Bat moves the eye point in the plane UN, see Figure 2.4.

2. **Simple Camera**. In this case the eye point is regarded as a video camera. Yaw of the Bat corresponds to rotation of the camera about the Z axis. Pitch of the Bat corresponds to the angle between the N axis and the XV plane. As before planar movement of the Bat moves the eye point in the UN plane, see Figure 2.5. Roll has no effect with this metaphor.

3. **Hand on Eye with Velocity**. This is similar to the hand on eye metaphor, except the planar displacement of the Bat is taken as velocity in the UN plane.

These metaphors are similar to Ware and Osborne's *eyeball in hand* and *flying vehicle* metaphors for navigation [WO90] in that they directly control the eye point with the input device. However because of the different designs of the input devices used, the manner in which orientation is controlled is different.

2.2.2 Object Positioning

Picking and moving objects is also a 6 degrees of freedom task, though here the situation is more complicated because the two tasks, picking and placement of objects, result in different demands on the metaphor used.

The Bat controls a 3D cursor and when an object is picked it becomes fixed relative to the cursor. Hence rotation of the bat causes the object to

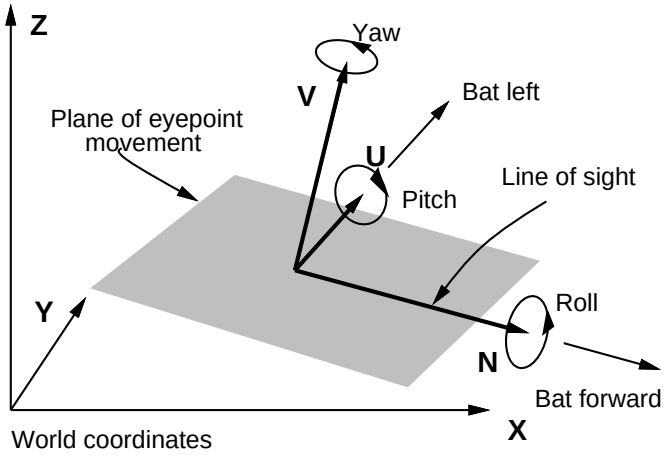


Figure 2.4: Hand on eye metaphor

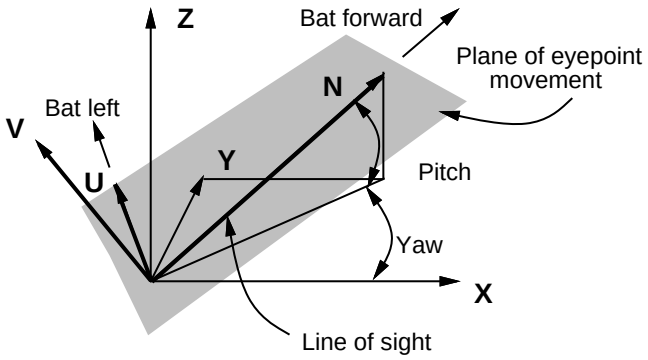


Figure 2.5: Simple camera metaphor

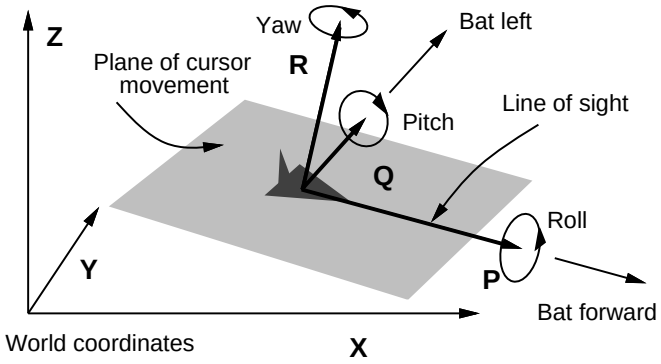


Figure 2.6: Relative to cursor metaphor

rotate about the cursor's centre. The cursor's coordinate system is defined by the axes P,Q and R where P is the direction in which the cursor is pointing and R is the cursor's up vector. The metaphors for moving the cursor are:

1. **Relative to Cursor**. Rotations of the Bat cause rotation of the PQR system. For example roll of the Bat causes roll of the Q and R axes about the P axis. Planar movement of the Bat causes movement of the object in the plane PQ, see Figure 2.6.

2. **Relative to world**. Rotations of the Bat cause rotation of the PQR system. However when the Bat is moved forward the PQR system moves along the vector defined by the projection of the axis P into the plane XY, see Figure 2.7.

3. **Relative to Eye**. In this metaphor rotating the Bat causes rotation of the PQR system in the UVN system. For example yaw of the Bat corresponds to rotation of the PQR system around the V axis. Planar movement of the Bat then moves the object in the plane defined by U and N, see Figure 2.8.

The Desktop Bat's buttons allow the definition of simple gestures to effect changes in the interaction mode. The most important gesture uses a single button that acts as a clutch. The clutch disables the Bat so that it can be used in a relative mode, or so that the hand can be reoriented to a flat, comfortable position. The other required gestures, each of which uses two buttons, are: to swap between eye point and cursor control; to enable and disable object picking; to recall the cursor to a position in front of the eye; and finally to enable translations of the eye or cursor in a direction orthogonal to that usually allowed by the metaphor. For example, to move vertically instead

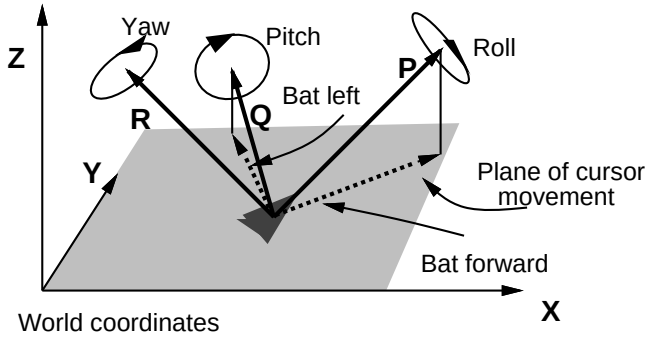


Figure 2.7: Relative to world metaphor

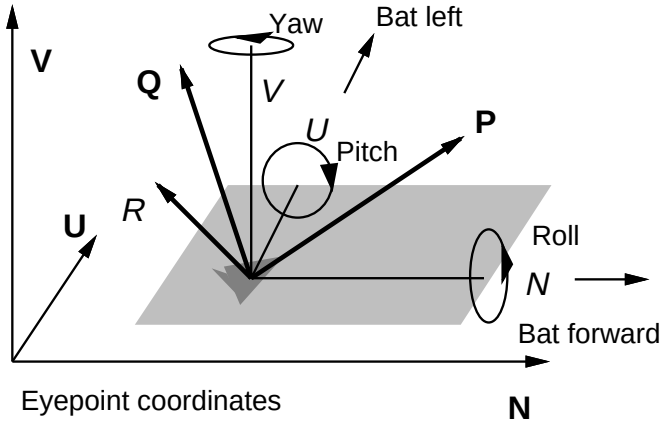


Figure 2.8: Relative to eye metaphor

of horizontally. The gestures were each designed with a simple metaphor in mind. For example the gesture to pick objects uses the buttons under the thumb and first finger, which corresponds to a pinching action.

There are nine possible combinations of these navigation and manipulation metaphors. Their relative utility for different applications is an empirical question and the next section describes an experiment designed to answer this.

2.3 Experimental Design

Different applications place different emphasis on the use of the input device for navigation, picking and placement of objects. Accordingly three scenarios were designed each with different primary tasks for the user to perform:

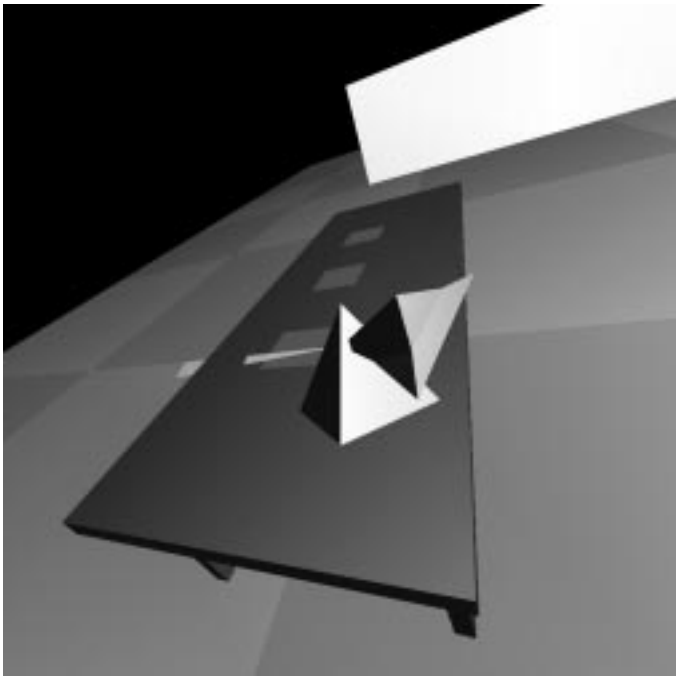


Figure 2.9: *Find* experimental environment

1. **Find**. Involved the user searching for three objects on a plane and then placing them on a table accurately enough to lock them into position. This places emphasis on the ability to *place* objects with the Desktop Bat. Figure 2.9 shows a user placing an object onto the table.

2. **Data**. This scenario consisted of a wire frame cube inside of which there were several variously shaped and coloured polyhedra. The user had to remove each of the eight yellow cubes from the wire frame cube, testing the ability to *pick* objects. Figure 2.10 shows a user about to pick a yellow cube.

3. **Maze**. This scenario consisted of three tunnels leading away from a central room. The task was to navigate along each tunnel to a room at the other end, and in that room to pick up the object resting on the table and to bring it back to the central room. This scenario was primarily to judge *navigation* ability. Figure 2.11 shows a user exiting a tunnel into the central room.

To evaluate the metaphors several measurements were taken automatically during the experiments. The efficiency of the navigation metaphor is indicated by the *Eye Time*, the time spent manipulating the eye; *Eye*

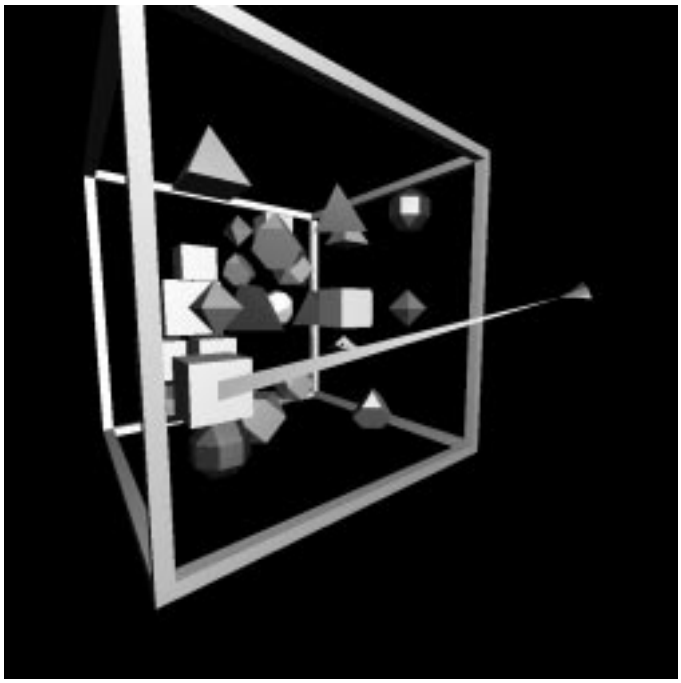


Figure 2.10: *Data* experimental environment

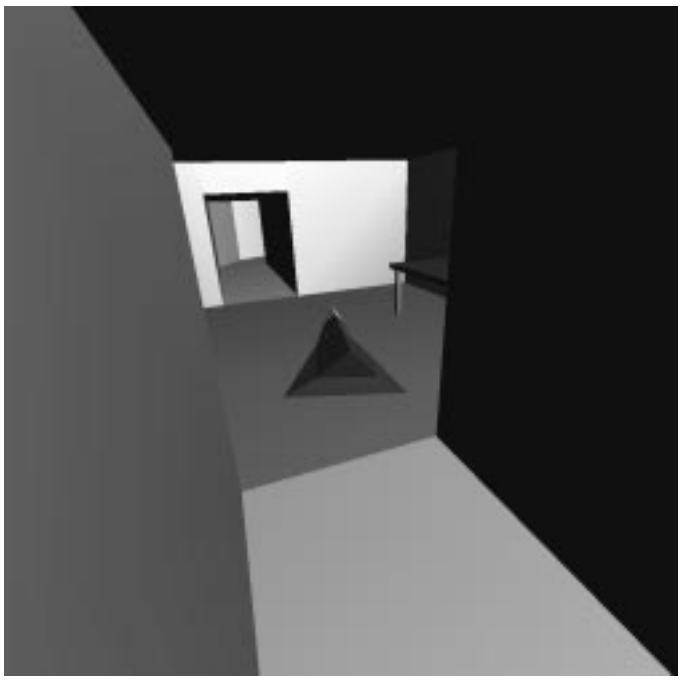


Figure 2.11: *Maze* experimental environment

Changes, the number of times the clutch was pressed when in eye mode; and *Eye Adjust Time*, the time spent using the clutch whilst in navigation mode. The overall efficiency of the cursor metaphor is shown by *Cursor Time*, the time spent manipulating the cursor; *Cursor Changes*, the number of times the clutch was pressed when in cursor mode; and *Cursor Adjust Time*, the time spent with the clutch depressed whilst in cursor mode. The times spent picking and placing objects were also of interest, and these are recorded as *Hold time* and *Pick time*. The number of times each gesture was used and the length of time spent using the mode which that gesture enabled were also recorded to see if the metaphors had an effect on them. These are: *Number of Recalls*, the number of times the cursor was recalled in front of the eye; *Cursor-Eye Mode Swaps*, the number of mode changes between navigation and cursor control; *Cursor Vertical Move Time*, the time spent moving the cursor vertically; and *Eye Vertical Move Time*, the time spent moving the eye vertically. Finally *Total Time* taken to complete the task was recorded.

The standard analysis of variance model for each of these variables was assumed:

$$y_{ijkl} = \mu + \alpha_i + \beta_j + \gamma_k + \delta_{ij} + \epsilon_{ijkl}$$

Where y_{ijkl} is an instance of one of the dependent variables, α_i the effect of being in the i^{th} eye metaphor, β_j is the effect of being in the j^{th} cursor metaphor, γ_k is the effect of being in the k^{th} scenario, δ_{ij} is an interaction effect between the eye and cursor metaphors, and $\epsilon_{ijkl} = Normal(0, \sigma^2)$. The index l refers to the repetition number for each combination of scenarios and metaphors and μ is the general mean.

A Latin Square distribution of metaphors and scenes was devised. Forty-five people took part in the experiment, allowing five repetitions of each scenario/metaphor combination. The subjects were recruited from amongst the students and staff of Queen Mary and Westfield College's Computer Science Department and Computer Services Centre. All had experience using a mouse-based computer interface. The system was implemented on a Silicon Graphics IRIS Indigo Elan using the GL graphics library.

The hypotheses of the experiment were as follows:

1. The *velocity control* metaphor for the eye would be best in terms of time taken to complete navigation tasks.

2. The *relative to cursor* metaphor would be best for picking up objects.

3. The *relative to eye* and *relative to world* metaphors would be superior to the *relative to cursor* metaphor for placing objects.

Eye Metaphor	Dependent Variables		
	Eyepoint Time	Eye Adjust Time	Hold Time
Hand On Eye	361.5 ± 196.4	74.6 ± 67.8	122.0 ± 62.2
Simple Camera	360.3 ± 196.3	73.5 ± 53.7	184.9 ± 148.8
Hand on Eye with Velocity	344.6 ± 218.9	36.0 ± 35.3	147.0 ± 127.9

Table 2.1: Mean and standard deviations in seconds of dependent variables for each eye point metaphor

4. Overall the *relative to eye* metaphor would be best for the general task of object manipulation.

2.4 Results

In the following analysis of the results in the first instance an analysis of variance (ANOVA) was performed on the data. From the results of this analysis the significant factors were determined and these factors were analysed further. The scenario will be a significant factor for each dependent variable because of the different tasks performed in each case and as this is not relevant in the comparison of the metaphors it will be ignored.

The results show that the *velocity control* metaphor is best for navigation and overall that the *relative to eye* is best for object manipulation.

In the following description, dependent variables for which there were no significant factors, except those directly relevant to the hypothesis, are not presented.

2.4.1 Eye Time

No factor was significant in the ANOVA. In particular the eye metaphor was not significant contrary to what was expected, as shown in Table 2.1.

2.4.2 Cursor time

The mean times by cursor metaphor shows what was expected, but the difference in the means was not statistically significant, see Table 2.2.

Cursor Metaphor	Dependent Variables: Cursor Time	Eye Adjust Time	Cursor Adjust Time
Relative to Cursor	229 ± 208	78.2 ± 65.0	8.8 ± 9.9
Relative to World	238 ± 136	70.2 ± 58.6	13.2 ± 12.6
Relative to Eye	163 ± 120	58.6 ± 31.4	6.0 ± 7.4

Cursor Metaphor	Pick Time	Hold Time	Cursor Vert. Time
Relative to Cursor	49.0 ± 77.6	180 ± 154	10.1 ± 18.4
Relative to World	70.2 ± 73.6	168 ± 109	56.0 ± 61.1
Relative to Eye	57.8 ± 90.6	105 ± 62	24.5 ± 20.4

Table 2.2: Mean and standard deviations in seconds of dependent variables for each cursor metaphor

2.4.3 Eye adjust time

In this case both independent factors and the interaction effect were significant at the 1% level in the ANOVA. The mean times for each eye point metaphor, shown in Table 2.1, suggest that the subjects spent less time adjusting the eye when using the *velocity* metaphor. The mean times for each cursor metaphor, shown in Table 2.2, are interesting as they indicate that the subjects spent significantly less time adjusting the eye if they had the *relative to eye* cursor metaphor. The reason for this may be that with the *relative to eye metaphor* the subjects did not need to move the eye point to a specific position. For picking objects it was advantageous to be level with the object in the *relative to world* metaphor. For placing objects it was advantageous to be above the placement area and looking almost directly down at it. With the *relative to eye* metaphor there is little advantage in such accurate placing of the eye.

The significance of the interaction effect is harder to understand. To solve the linear model most components in the interaction effect were aliased to zero. However the resulting model tells us that when using the *simple camera* metaphor for the eye, the best accompanying cursor metaphor is the *relative to cursor* with the other two cursor metaphors having little to distinguish between them.

2.4.4 Cursor adjust time

Cursor metaphor was significant in the ANOVA at the 10% level. The mean times for each object metaphor, shown in Table 2.2, suggest that the *relative to world* metaphor requires most use of the clutch and the *relative to eye* requires least.

2.4.5 Eye changes

Both independent factors and the interaction effect were significant in the ANOVA. The analysis of times provides the same relationships between the metaphors as those for eye adjust time as would be expected, the number of times the clutch is used and the actual amount of time spent using it are directly related.

2.4.6 Pick time

No factor was significant in the ANOVA. However looking at the means for each cursor metaphor, shown in Table 2.2, indicates what was expected.

2.4.7 Hold time

Both independent factors and the interaction effect were significant in the ANOVA. Cursor metaphor was significant at the 1% level, eye metaphor at 5% level and interaction effect at the 10%. The results show what was expected for the cursor metaphor, as shown in Table 2.2. The mean times for each eye metaphor, (Table 2.1), indicate that the *hand on eye* and *velocity* metaphors give an advantage over the *simple camera* metaphor.

The model also suggests that if using the *simple camera* metaphor for the eye, the best accompanying cursor metaphor is the *relative to world*, followed by *relative to eye*, then *relative to cursor*.

2.4.8 Cursor vertical move time

Interaction effect and cursor metaphor were both significant at the 1% level in the ANOVA. The mean times for each cursor metaphor, see Table 2.2, show that the *relative to world* metaphor required the vertical mode to be used frequently because it is necessary in order to change the horizontal plane in which the cursor moves. The *relative to eye* metaphor also requires the vertical mode to pick or place an object not in the plane of the eye.

The model also suggests that if using the *simple camera* metaphor for the eye, the best accompanying cursor metaphor is the *relative to world*, followed by *relative to cursor*, then *relative to eye*.

2.4.9 Eye vertical move time

Scenario and interaction effect were significant in ANOVA, interaction effect at the 1% level. The results for the interaction effect show that when using the *simple camera* metaphor for the eye then the best cursor metaphor to combine it with is the *relative to world*, then the *relative to cursor*, and then the *relative to eye*.

2.5 Conclusions about the Desktop Bat

The results are consistent with the hypotheses, but for reasons not always anticipated.

The *velocity control* metaphor is best for navigation, though this is not due to the actual time spent moving, but the lesser amount of time spent using the clutch with this metaphor. This is understandable since to navigate long distances with the *simple camera* or *hand on eye* metaphor requires the clutch to be used whenever the bat has been pushed as far as the arm will reach. When using the *velocity control* metaphor the clutch was only used when turning a corner around which the subjects could not see, in which case they would stop, turn and then speed up again. The *velocity control* metaphor also has an advantageous effect on the time spent holding an object, though the reason for this has not been determined.

For the object manipulation metaphor the results have confirmed that different metaphors are best for the two tasks of picking and placing. The *relative to cursor* metaphor was marginally better for picking objects when looking at the mean times for object placement, and the *relative to eye* was best for placing objects. The *relative to world* metaphor did not give as much of an advantage for placing objects as originally thought. Overall the *relative to eye* metaphor would be the best choice due to the greater difficulty of placing objects compared to picking them. This is confirmed by the analysis of the cursor adjust times and cursor changes that would seem to suggest that orienting the cursor is easier with the relative to eye metaphor. The cursor vertical move time indicates that the *relative to eye* metaphor requires a lot of vertical movement, but this time is a part of the total time spent using the cursor and so does not suggest a poorer overall performance for this metaphor.

More succinctly, the *velocity control* metaphor is best for navigation and overall the *relative to eye* is best for object manipulation.

2.6 Limitations of the Desktop Bat

The Desktop Bat has advantages over some other devices in that it rests on the desktop and does not lead to fatigue from having to hold out a 3D position sensor. It also can be used as a normal mouse and my be especially useful for applications that require switching between 2D and 3D environments. The lack of a sixth degree of freedom is not a problem since the metaphors discussed earlier allow general movement in 3D environments. In fact object placement may be easier if not all the degrees of freedom of a 6 degrees of freedom device are enabled simultaneously [War90].

However, there are some drawbacks especially when compared to the possibilities provided by IVEs. Firstly mode switching and overloading are inevitable since one device is being used for multiple tasks. Secondly interactions should ideally be customized for user preference. And finally, none of the interaction metaphors are naturalistic since they require some cognitive mapping from physical action to result within the environment.

The use of one device for several tasks, and the multiplicity of interaction metaphors for different tasks means that the interface will have to include switches to determine which mode the device will operate in, thus overloading the device by giving it more than one function in the environment. As we have seen there are at least two distinct modes, navigation and object manipulation, but further sub-modes might be useful. With suitable choice of metaphor we shall see in Chapter 3 that this mode-switching and overloading are avoidable within an IVE.

A limitation not specific to the Desktop Bat is one resulting from the fact that the environment is displayed on a desktop machine without a head-tracked display. This is that some spatial tasks are better performed within an immersive system rather than when watching "Through the Window" [SAU94].

A further conclusion drawn from the experiment is that interactions should be customized for the user. Ideally the option to use any of the metaphors should be left open to the experienced user, but even then further customization may be desired. One instance where this would have been useful during the experiment was in selecting the manner in which the clutch was used. The metaphors specified that the clutch disabled rotations and translations, but some participants expected the opposite. The participants who expected the clutch to enable rotations and translations used the clutch

in the same manner as a ratchet, and started moving in the opposite direction to that which they desired. The alteration to the system required was subtle and trivial to perform, though it was not anticipated prior to the experiment.

Finally when using the Desktop Bat the user has to learn news skills to perform tasks and from observation participants initially had to think in terms of how to move the cursor in the direction they want rather than using a skill they know how to use.

Overall, though this chapter has focused on a single Desktop Bat, we have seen that there are inherent limitations to exploring a 3D environment using a 2D interface. These limitation arise from the use of indirect and non-naturalistic metaphors for interaction and the poor sense of position that arise from NIVEs. Chapter 3 will show how appropriate metaphors for IVEs allow the user to use skills and techniques they already possess in order to accomplish similar tasks.

Chapter 3

Immersive Interaction

This chapter focuses on immersive virtual environment systems and the interaction metaphors and styles that they afford. The chapter shows how the semantics of the IVE can dictate appropriate metaphors for interaction and again it leads to the requirement that VEDA should not constrain the interaction techniques that can be described.

The basic aim of the IVE systems we will describe is to generate and sustain a sense of presence within the virtual environment they display. Thus we will discuss the sense of presence, how to measure it and the effects of various interaction metaphors on the sense of presence. The main work in this chapter is the investigation of a technique for navigating within IVEs called the "Virtual Treadmill". This is an example of a gesture based metaphor and the comparison between it and standard navigation metaphors for IVEs will demonstrate how using a model of the participant's body can make the interaction techniques more independent of the input devices. This body model thus relieves the problem of interaction techniques being tightly bound to input devices as was the case with NIVE systems as described in Chapter 2.

Section 3.1 outlines the basic elements, hardware and software, that are needed in order to generate an IVE. Section 3.2 describes the scope of gesture based interaction and a few systems that support gestures. Section 3.3 defines presence and lists some exogenous factors, that is aspects of the presentation and composition of the IVE, purported to affect it. These exogenous factors are examined further in Section 3.4 and this will lead to investigation of endogenous factors in Section 3.5. The problem of navigation in IVEs and it's affect on presence is discussed in Section 3.6. A specific interaction metaphor is introduced in Section 3.7 and examined with respect to it's affect on presence and performance in Section 3.8. Section 3.9 describes some extensions to the Virtual Treadmill metaphor and describes a general

approach to metaphor construction. Finally Section 3.10 draws together presence and interaction with a new model of presence and Section 3.11 concludes by indicating some requirements for the design of IVEs.

3.1 Immersive Virtual Environment Systems

There have been many attempts at defining what constitutes an IVE system. A common theme is that IVE systems are striving towards becoming what Ivan Sutherland called the 'Ultimate Display'[Sut65]:

> a system that can present information to all the user's senses at a resolution equal to or greater than that he or she can discern so there is no way to tell that the artificial world is not real.

Such a system would be the ultimate direct manipulation system (§1), in that if the artificial world is convincing it must be possible to act within that environment as if one were actually there.

A system that generates an IVEs provides the participant with a sense of *presence*, that is it provides an illusion that the participant is physically present in a place other than that where their real body is. This sense is what both Minsky [Min84] and Sheridan [She92a, She92b] termed *telepresence*.

Many different media forms have been claimed to give this belief to a certain extent. For example, films are often classified as good or bad depending on whether or not the viewers felt 'there' and had observable reactions to the on screen action. On this criterion we should include such media as video, computer games, photography, compact discs and so on in our definition of IVE systems. Warren Robinett [Rob92] proposes a taxonomy that covers the whole breadth of technologically mediated experience.

However there are subtle distinctions to be made between presence and psychological states such as awareness, attention and focus. In general a person who is present in an environment describes that environment as somewhere they have visited rather than somewhere they have seen. A fuller explanation of presence, it's definition, measurement and utility in evaluating interaction metaphors for virtual environments is given in Section 3.3. As an example though, one participant who experienced a virtual environment [SU92] made the following comment:

> My feeling when carrying on with the experiment was that of being in another part of the building where the experiment was held...

One of the first systems whose purpose was to provide the participant with a sense of presence was created by Morton Heilig with his 1962 'Sensorama Simulator' [Rhe91, pages 49-53]. This was designed as an arcade machine that would provide the sensation of riding a motorcycle through Brooklyn. It provided an experience where the participant saw a 3D movie of a motorcycle ride complete with engine vibrations, wind and smell effects.

The major difference between Morton Heilig's system and current systems is in the fact that the Sensorama machine was non-interactive whereas today the emphasis is on systems that provide the participant with control of the experience and the ability to control it.

Myron Krueger's various 'Artificial Reality' systems do provide the high degree of interaction that is missing in Morton Heilig's Sensorama[Kru90, Kru91]. Krueger's fundamental concept is that of an environment that responds to the participant's movements. In his system VIDEOPLACE the participant enters a darkened room and is confronted with a wall-sized display whose image is their own silhouette. A hidden computer can add graphic objects to the display and the participant can interact with these objects in a multitude of ways. Many scenarios have been created, one of the most amusing involves a CRITTER, a small graphic creature which with the correct coaxing will jump on to an outstretched hand, dangle from a finger or perform a jig on the participant's head. Whilst this scenario is for entertainment there are serious applications for this system, one of which is remote conferencing as the images of several environments can be combined to allow the participants in their individual environments to interact in the same artificial space.

VIDEOPLACE also differs from most current systems in the fact that the participant sees a third person view of himself within the computer generated environment. An immersive system would usually display a first person view which was slaved to the participant's movements.

A virtual environment generator, or *virtual reality* system as has been popularized by the media over the past few years has come to mean a computer system such as described in [FMHR86, Fis90, SIG89, SIG90, Bri93] that has several key components:

- A database that describes the VE. As detailed in Section 1.1, this describes the content, geometry and dynamics of the VE, and includes a special object that is the representation of the participant, the "self" within the VE.

- A head mounted display (HMD) that provides a visual representation of the VE, rendered from the viewpoint the participant within the

VE. A HMD usually uses twin displays, one for each eye, and the images for each are produced using perspective projections with slightly different centres of projection for each eye in order to give stereo cues [Sut68, CHB+89, RR92, Pat92].

- Audio output to complement the visual images. This aspect receives less attention than the visuals of an environment, though techniques to create and locate sounds in 3D space do exist and are very convincing [Wen92, DRP+92].

- A system that can track the participant's body [MAB92]. This is usually restricted to tracking the hand and head, in order to provide a viewpoint from which to render the VE, and the second to an effector within the world with which to interact with objects.

- A gesture input device that the participant holds in or wears on his hand. This can be as simple as buttons built into the hand held tracking device, or a glove that tracks hand posture.

However an IVE system is more than just a description of a technology, the VE itself must appear to be consistent and provide a degree of interactivity. One categorization of VEs that emphasizes the nature of the environment rather than the technology is the Autonomy, Interaction and Presence (AIP) cube proposed by David Zeltzer [Zel92], see Figure 3.1. An IVE system needs to be interactive, that is the participant is able to change events and not just be a spectator as in Sensorama. An ideal IVE system should provide a high degree of presence so that the participant would feel that they were in the world created by the computer. It would have to be autonomous, that is the virtual world would not be a passive structure, but should have the ability to react to a wide variety of stimuli.

Section 3.1.1 will discuss the basic devices that the systems will use and Section 3.1.2 will discuss metaphors for interaction within virtual environments.

3.1.1 Immersive Virtual Environment Devices

A complete review of IVE devices is beyond the scope of this thesis, the reader is referred the books of Burdea and Coiffet, Barfield and Furness, and Kalawsky, as excellent starting points for investigation[Kal93a, BC94, BF95]. This section will cover a number of example technologies to illustrate the basis from which we have to work when constructing interaction techniques.

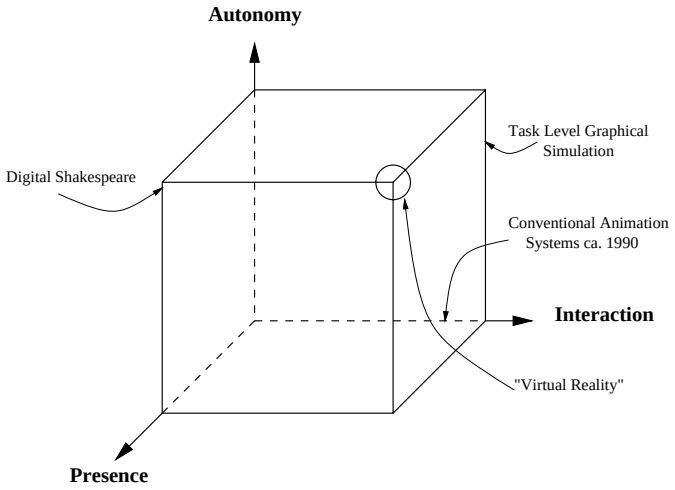


Figure 3.1: Zeltzer's AIP cube with example systems

An example display device, the one used in the work described in the rest of this thesis, is the Virtual Research Flight Helmet shown in Figure 3.2. It uses twin back lit liquid crystal displays, both having a 360x240 colour resolution. The field of view is 100^o degrees in the horizontal and 60^o degrees in the vertical. The device uses a NTSC signal and weighs about 2kg.

Although a relatively old device, it is typical of the resolution of most of the head mounted displays in use today. Head tracking is provided by a Polhemus Fastrak system (§2.1).

In an IVE system the participant's hand is usually tracked by a second device. The Division 3D mouse, see Figure 3.3, is a typical device. The tracker is embedded within a pistol grip that has five buttons, three on the top to be used by the thumb, and two to be used by the first and second fingers.

Although a glove device was not used in the work in this thesis, they are quite commonly used and have been one of the icons of virtual reality in the media [Fol87]. The VPL Dataglove [ZL87, ZL91] (§2.1), works by measuring the leakage of light from optical fibres stretched over the finger joints. From the loss of light a crude measure of angle can be obtained, though the Dataglove was notorious having to be recalibrated for each participant and each session [BC94, pages 34-35].

3.1.2 Interaction Techniques

Interaction within an IVE differs from interaction with a NIVE in a number of important ways:

Figure 3.2: Virtual Research Flight Helmet

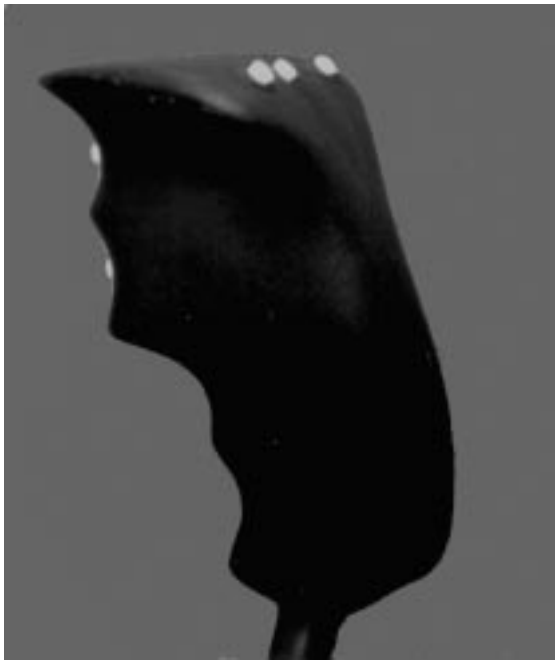


Figure 3.3: Division 3D Mouse

1. The participant is immersed with the virtual environment and to some extent is unaware of the physical environment around them, in particular the display and sensing devices.

2. The whole body is involved with interacting with the virtual environment and in particular the participant is usually standing.

3. The participant can move freely and a number of these movement are captured without being constrained by the number of degrees of freedom of the interaction devices.

The interaction devices thus return a partial record of the state of the participant's body and the participant's representation within the environment is usually consistent with this record, rather than being an abstract viewpoint and detached cursor.

These differences provide an environment within which it is supposed that certain tasks can be performed more easily than on a desktop system. One example is architectural walkthrough where experiencing the virtual environment from a realistic perspective gives a sense of being within that environment that is similar to being within the actual space it portrays [SAU94].

Direct manipulation is the basis of IVE systems, with most systems representing at least one of the participant's hands inside the environment which can be used to interact with the objects in a natural and realistic manner.

However IVE systems still require navigation techniques because, although they are unlike NIVE systems in that the participant is free to move around, there is a limitation to the distance they can physically move because of the range of the interaction devices.

Two typical navigation metaphors are:

1. Fly in the direction the hand is pointing by making a gesture such as pressing a 3D mouse button.

2. Fly along line of sight by making a gesture.

Flying in the direction of hand pointing is a useful metaphor as it allows the participant to fly around an object while their gaze remains fixed upon it. Unfortunately it can be tiring for the participant's to continually have to point where they want to go, so flying along the line of sight, which doesn't allow the same flexibility in direction but does allow the participant to leave their arm at their side, is a possible alternative. When flying along the line of sight the hand is free to hold objects, and to hold them so they don't obscure the view which can be a problem with the fly in the direction of pointing metaphor if the participant wants to fly forward but is holding a large object.

3.2 Gesture Based Interaction

Gesture recognition can be performed on any tracking device that senses the position of a body part, though a greater breadth of gestures should be possible when more tracking devices are used. With an IVE system, where we have a direct correspondence between body position and representation within the environment, the potential for gesture based interactions is great since the participant can make natural gestures, such as those they make in the real world.

The simplest type of gesture interaction is the symbol set, where the system recognizes a number of defined postures of the participant. In general though gestural interaction is much broader than symbol sets and the following section describes a classification of gestures. Detecting when the participant is making a particular gesture is the concern of Section 3.2.2 and Section 3.2.3 describes some systems that rely on gesture based interaction.

3.2.1 Gesture Classifications

The AHIG system is one of several categorizations of gestures [Wex94]:

- symbolic/modalizing
- pantomimic
- iconic/object
- deitic/Lakoff
- beat/Butterworth/self-adjusters

Symbolic/modalizing These are culturally defined gestures such as thumbs up and V-for-victory that are purely symbolic, that is the physical postures and motions bear little or no relation to the gesture's meaning.

Pantomimic These gestures mime the use or action of an object. There are many examples, though their meaning may not be immediately obvious (consider playing a game of charades) and depends on context. An example might be miming opening a stuck door.

Iconic The body becomes the object being described. For example a person might describe the orbit of a satellite around the earth by making one fist stationary and making the other one circle.

Deitic/Lakoff These gestures indicate items of interest in the environment. Pointing, nodding at or gesturing with the back of the hand are all examples of this type of gesture.

Beat/Butterworth/Self-adjusters Beat gestures mark stress, rhythm and pace usually of speech, though an orchestra conductor uses the same techniques. Butterworths also serve as marks, though they indicate that the exposition is not complete or the speaker is thinking. They do not generally convey any information about the subject being explained. Self-adjusters are fidget gestures that like Butterworths do not convey information about the speakers subject but about their mental state.

All these types of gesture have their application in three dimensional systems, though by their nature some are more difficult to recognize than others. Many symbolic gestures such as the two mentioned are easy to recognize provided the fingers and hand are tracked since they have a static component (the finger shape) that does not depend greatly on hand position (though orientation is significant for "thumbs up"). Recognizing dynamic gestures, where the limbs move over time, is more complicated, though their expressive power is greater.

3.2.2 Gesture Recognition Software

The problem of gesture recognition is to classify motions of the body into events inside the virtual environment. This problem has been tackled for pen based input systems [Lip91, Rub91, Zha93] but as Wexelblat notes [Wex94] in an IVE system the problem is harder as we don't have events to mark out the gesture in time (the pen-down and pen-up events).

Kendon [Ken80] postulated that gestures consist of five phases: preparation, pre-stroke hold, stroke, post-stroke hold and retraction. Unfortunately gestures can run together and Kendon admits that only the stroke phrase is necessary so there is a problem partitioning the input data into different gestures. One system that uses a form of preparation is the Charade system [BBL93] for controlling a hypercard presentation. Gesture recognition is only activated once the hand is pointing towards an active zone, otherwise the presenter is free to leave their hands by their side. Sparell's systems [Wex94, pages 36-37] use a three phase approach: gesture recognition is enabled as the participant moves their arms up from their sides and finishes as they return them. These approaches hinder the natural expression of gesture, and ideally gesture recognition should be continuous and freely directed.

Current techniques for recognizing gestures include explicit recognition, template recognition, feature based recognition and neural net based recognition [Wat93b].

Explicit Recognition In these systems for each gesture to be recognized there is a specifically coded procedure that detects some feature of the gesture.

Using a dataglove as input Sturman uses explicit software formulations to detect the dedicated set of gestures for his example applications [Stu92]. For example to recognize a waving gesture the features are: hand not closed, more than a fixed number of direction changes of the fingers in a set period and waving motion large enough not to be random fluctuations. Sturman does suggest that further work should concentrate on extending the set of features to cover a majority of useful gestures.

Template Recognition Template recognition involves matching the current state of the raw input data to a set of ranges that correspond to gestures. This will often involve a least distance estimate to distinguish between likely candidates. At its' simplest level this involves matching the bend values associated with the finger joints to a predefined set of bend values that model the posture to be matched.

The VPL system described in [ZL87] used this method. It was extended to incorporate hysteresis values to widen the required range so that once a gesture was first recognized holding it would be easier for the participant.

The MR Toolkit also used a similar approach, but incorporated a gesture editor where the range of values for each joint can be specified in a window or by demonstration [Kal93b, page 231].

Sequences of such postures can be used to recognize dynamic gestures. Bordegoni *et al.* use a sequences of postures along a particular trajectory for dynamic gestures [BH93]. Their system also provides a gesture editor that illustrates the gesture with a 3D model of the hand and the path and poses it makes.

Lipscomb uses templates for pen based recognition [Lip91]. The sequence of points produced is compared against a hierarchy of templates which increases with complexity from broad forms at the bottom to specific gestures at the top. This reduces the work involved with comparing the raw data with all possible templates.

Feature Based Recognition To overcome the complexity of recognizing a gesture from many possible templates, features can be extracted from the

raw data and these feature vectors classified.

Rubine [Rub91] built a handwriting recognition system for a single point on a plane, either a pointer, mouse or stylus. As noted before, the start and end of the gesture are marked by the start and end of the list of points returned by the hardware. The features extracted from the list of points include distance between first and last point, initial angle and diagonal angle of the bounding box.

Some of these features are not appropriate for continuous 3D gesture recognition and Rubine gives a methodology for 3D, but still with the constraint that the start and end points must be specified.

Sturman [Stu92], uses a small set features as the basis of his gesture recognition. Typical features are path bounding volume, cumulative path length and current linear speed.

A simpler method of choosing the features was chosen by the GLAD-IN-ART project [Wat93b] They choose the features to be discontinuities of the raw input, either turning points or the beginning and end of constant regions in the individual data streams. Wexelblat [Wex94] uses a similar approach as the first stage to segment the data values but the interpretation of the path produced by the feature detector is provided by a separate context dependent module that makes the overall system more flexible.

Neural Net Based Recognition Neural networks [HKP91] are an obvious choice for gesture recognition because training is done by giving examples to the network and because of their ability to generalize and their resilience to errors in the input data. Unfortunately training can take a long time and once the network is trained it can not be modified to recognize another gesture without retraining the whole net.

The Glove-Talk system [FH93] uses five neural nets to recognize hand gestures and drive a speech synthesizer. The language consists of a set of hand-shapes with the direction indicating the word ending and stress and speech rate determined by the duration and magnitude of the gesture.

Murakami *et al.* used a different type of neural network for their dynamic gesture recognition [MT91], to recognize ten symbols from Japanese sign language. The recurrent network they used only had data for the last 3 time steps as input but maintained context by using the values for the hidden layer in the previous time step as input. Training for the 10 gestures then took 4 days on a SUN/4 workstation to achieve a 96% success rate.

Recognizing gestures for an IVE system has complications due to the nature of the interface devices used, see Section 3.1.1. Current tracking systems

have to trade off between resolution, lag, working area and cost. That the data is returned with some lag is not necessarily a problem in itself for the recognition software, but since recognition techniques themselves take time to run, the overall delay between an action being performed and its recognition can be quite significant. Another problem that affects magnetic systems in particular is that not only can the data be noisy, but the noise level might depend on the distance from the trackers and the underlying non-noisy position may be distorted due to magnetic properties of other objects in the environment. This makes simple template based recognition unreliable since hysteresis values must be set quite high unless the participant's freedom of movement is further constrained. Neural net techniques might be more appropriate because of their resilience to noise and Section 3.7 describes the use of such a net to recognize a specific gesture.

3.2.3 Gesture Based Systems

IVE systems often use a single glove-based with simple gesture recognition to activate tools and modes of the interface. With VPL's RB2 system [BBH+90] or the GIVEN toolkit [BHV92] a first finger pointing gesture enables unconstrained flight around the environment, whereas a flat hand enables flying in the vortex world designed by Lewis *et al.* [LKL91].

The range of gesture used in virtual environments is broad, though there is a general design criteria for the sets be natural and easy to remember. This involves making the gesture pantomimic or the use of well-known symbolic gestures, such as thumbs up for confirmation, rather than introducing arbitrary symbolic gestures such as the flying gestures above. This rule is usually sufficient, given the limits of current gesture recognition algorithms, to cover *mundane* actions inside a virtual world, but could easily fall down when gestures for the *magical* abilities of the participant in the virtual world are considered, such as scaling and more general data manipulation.

A particular limiting factor is that most gesture sets are based around the participant's hand shape, with hand position and orientation only significant for certain dietic gestures where the hand indicates a selection object. This limits the gestures that can be accommodated, and we shall see in Section 3.7 how a full body gesture provides a natural metaphor for navigation. Indeed since the hand is the only input device in some applications their can be a tendency to overload the hand with abilities. In a CAD based system described by Weimer and Ganapathy [WG89], the thumb abduction gesture had three meanings dependent on context: picking objects, enabling a clutch for incremental transformations, and throttling where the abduction angle is used as a scaling value.

3.3 Presence

Presence in an IVE system would appear to differ from the sense of absorption, engagement or suspension of disbelief that may arise from a book or game in that after the experience many people describe the virtual environment as a place they have visited rather than an environment they have seen. Some comments made by participants illustrate this [SU92]:

> Looking back it feels more like somewhere I visited, rather than something I saw (as in a film), so I suppose I must have felt I was in the scene.

> In fact the 'virtual reality' world was more real than I was expecting. I had the impression I was in a real room...

This sense that the person has of being in an environment other than that where their real body is similar in concept to what both Minsky [Min84] and Sheridan [She92a, She92b] termed *telepresence*.

Many factors have been suggested that increase the sense of presence [She92b, Loo92a, Loo92b, Hee92, SU92, SU93, Ste92, BH95, BSZS95]:

1. The data presented to the senses should be of high resolution.

2. The data should not be obviously from an artificial source. For example the displays should be refreshed at a rate high enough so the participant does not see flicker and the displays themselves should not be so heavy that this becomes a source of fatigue.

3. The data presented should be consistent. For example if an object in view makes a sound then the sound should appear to originate from that direction.

4. The virtual body or slave robot should be similar in appearance to the operator, so there can be an identification between the participant's limbs and those of the representation.

5. There should be a direct visual consequence of each of the participant's movements.

6. There should be a obvious mapping between the participant's movements and the movements of the virtual body or slave robot.

7. There should be a wide range of possible interactions that the participant can make. For example if there is a virtual table in the environment then you should be able to not just see it, but touch it and feel its weight.

8. Other objects or participants in the environment should recognize and acknowledge the participant in some way (such as a door opening as the participant approaches).

These factors are all *exogenous*, that is they concern the presentation and behaviour of the environment and are external to the participant's body.

The above criteria do not specify natural behaviour for any object, only that there should be a direct relation between efference and afference. Of course in a real world simulation experience, such as an architectural walk through application, natural behaviour of objects should be our aim but not at the expense of consistency. Held and Durlach [HD92] and Loomis [Loo92a] note that the operator's sense of presence can increase over time, which could be due to the participant's coming to understand the world's causal laws and gaining a belief in the consistency of the environment model.

3.3.1 Measuring Presence

To determine the suitability and effectiveness of changes to the virtual environment we need a working measure of presence, but as Held and Durlach, and Sheridan note none currently exists. Methods do exist for measuring psychological states such as involvement and engagement, but it is believed that presence is a different state. Suggested approaches for measuring presence [SU92, BSZS95] include:

1. Participant's reported sense of presence. This is a complicated process because the process of enquiring the state of the participant may change that state.

2. Observations of the participant's behaviour. This takes observable reactions to certain situations as confirmation of the participant's presence. For example shying away from looming objects or replying to a welcoming 'hello' message.

3. Performance of tasks in real and virtual environments. This assumes that if a participant performs a task in a virtual environment as efficiently and in the same manner as they do in a real environment then they must be present in that virtual environment. This however

would work only for naturalistic environments and is of more use in the teleoperator field.

4. Discrimination between real and virtual events. This tests, for example, the participant's ability to differentiate between sound cues that originate within the virtual environment and those that originate in the real world.

5. Incorporation of external stimuli. If the participant interprets an external event, such as a loud noise, in the context of the virtual environment then they must be present in that virtual environment.

All but the first method present immediate problems, in that taking the measure could either break the metaphor for the environment or actually present stimuli that might affect the sense of presence.

The following section investigates the relation between these measures and will lead in Section 3.5, to our having to include considerations about endogenous factors of the participant when using their reported sense of presence.

3.4 Exogenous Factors

One of the factors that was suggested to increase presence was to provide the participant with a virtual body. In an architectural walk through application the virtual body is especially useful as it can provide useful clues to spatial relationships between objects since it provides the world with a natural scale.

Given the limited amounted of information about the participant's posture known by the system a number of approximations have to be performed [SU94b] in order to create the virtual body. Since the torso is not tracked explicitly, it was assumed to hang directly below the body, and only turn once the head position has turned by 60^o around the vertical axis. Similarly, since the hand was tracked and not the elbow, the arm was assumed to extend directly from the shoulder. And again since there would be no difference in reported hand and head position for a bend from the waist posture and crouch gesture, if the head dipped below a normal height the participant was assumed to be crouching. This normal height was the recorded height of the participant when they entered the environment, at which point they were assumed to be standing erect.

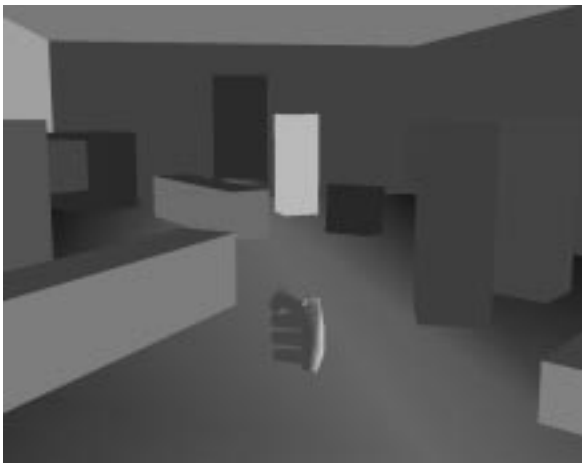


Figure 3.4: View from far end of cluttered room

3.4.1 Experiment

In a pilot experiment, conducted to test whether the virtual body increased the sense of presence and generate further hypotheses [SU92], the subjects were split in to two groups, one group had a virtual body and the other had a simple arrow representing the hand, and all went into six rooms in the virtual environment in turn.

The first room was cluttered with objects and the task was to navigate to the other end of the room. The hypothesis here was that those with the virtual body would make fewer collisions with objects as they would be more careful about avoiding them since they would have visual feedback if they collided with something. See Figure 3.4.

The second room had objects that flew towards the position of the subject's real body with the hypothesis being that those without the virtual body would show a lesser reaction.

The third room involved the subjects building a pile of blocks during which the virtual body would disappear for a short period. The purpose of this was to give all the subjects a lengthy task to complete to assess whether task involvement would lead to a higher sense of presence and also to see whether those with a virtual body would react to its disappearance.

The fourth room was similar to the second except that the objects approached the face.

In the fifth room the body was re-oriented so they would appear to be upside down. The purpose of this was to see the effect of the disparity between the subjects sense of orientation and the visual information presented.

The last room consisted of a chess board with a plank upon which was a

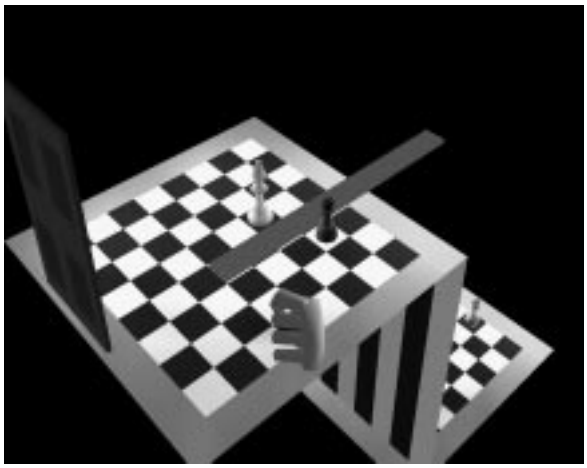


Figure 3.5: View from above down into room with plank over precipice

plank that led out over a precipice. Here we expected to observe a fear reaction, with a possible difference between those with and without the virtual body. See Figure 3.5.

17 participants took part in the experiment, with 9 in an experimental group having a virtual body and 8 as a control group.

The experiment was implemented on a Division Provision 200 system, using the DVS operating system. The display device was a Virtual Research Flight Helmet^{tm} with a resolution of 360x240 colour triads and field of view of 75 degrees along horizontal and 40 along the vertical. Rendering was performed by Intel i860 microprocessors with one per eye generating a RGB-170 video signal which is then converted to NTSC to drive the display. Tracking was provided by Polhemus Isotracks updating at 30hZ, one on the helmet and another inside a Division 3D Mouse.

3.4.2 Results

The two measures of presence taken were observed reaction to the approaching objects and plank, and reported sense of presence as gathered by a post-experiment questionnaire. With this group of people these did not correlate well - some people who had adverse reaction to being on the plank did not rate themselves as being present. It could be that the sense of presence varied over time and the subjects were reporting their overall impression.

The main interest was the influence of the virtual body on the sense of presence. From the direct results it was seen that those with the arrow representation reported a higher sense of presence than those with the body,

opposing the hypothesis. This could be explained by taking into account that those with a predisposition to travel sickness reported a higher sense of presence and there were more of these people in the no-body group. The analysis also showed that the non-body group contained far more people who considered themselves able to adapt quickly to new circumstances.

By considering various other personal factors the following tentative conclusions were drawn:

- Those without a virtual body who mentioned display problems were likely to report a low sense of presence. There was no difference between those with a virtual body.
- In the group that had virtual body the females generally reported a higher sense of presence than the males. The reverse was true for the other group.

Overall the virtual body was seen to have an influence, but in a complicated manner [SU92, SU93].

3.5 Endogenous Factors

The experience with the first analysis lead directly to the use of a model of the subjects' psychology in order to fully understand the effects of the virtual environment.

The approach taken was to use the unorthodox Neuro-Linguistic Programming model [DGB+79]. This classifies people along two axis, *representation system* and *perceptual position*. The representation system determines whether the persons' dominate mode of thinking is visual, auditory or kinaesthetic. That is, do they think in terms of pictures, by internal-dialogue, or in terms of sensations and emotions. The perceptual position is the standpoint from which the person experiences and remembers events. This is either first, second or third person. That is they remember events from either their own perspective, that of another person or from a disembodied viewpoint.

The representation system and perceptual position of each of the subjects was determined from the last part of the questionnaire, which asked subjects to write about their experience, by counting the number of visual, auditory and kinaesthetic predicates and references used. For example:

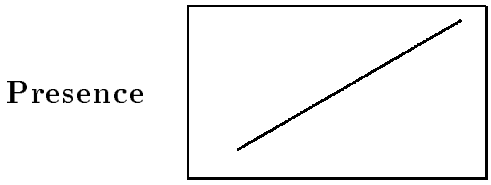
> In many of the rooms I visited, I felt I was really in that world

would be classified as first position with a kinaesthetic predicate.

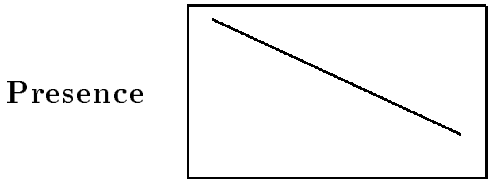
3.5.1 Effect of the Virtual Body

The outcome of analyzing the results from the original pilot experiment using the NLP technique was much more productive, see Figures 3.6(a,b,c) [SU93, SU94b].

(a) Variation of Presence with Visual Dominance



(b) Variation of Presence with Auditory Dominance



(c) Variation of Presence with Kinesthetic Dominance

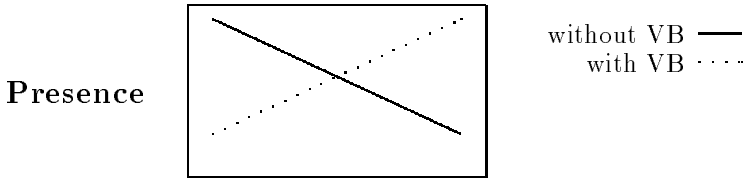


Figure 3.6: Variation of presence with representation system

To summarize these results:

- If a subject was more visually dominant then their reported sense of presence was higher, regardless of whether they had a virtual body.
- If a subject was more auditorily dominant then they were less present.

- For those with a virtual body, the more kinaesthetic they were the more present they were. For those without a virtual body the opposite was true.

- The level of presence increases with the first perceptual position up to a certain point then decreases.

The interpretation of these results is quite intuitive. Since with the ProVision 200 system used in this experiment the experience is primarily visual, the more visually orientated a person is the more present they are. There is almost no sound in the environment used for the experiments so it follows that a auditorily dominated person might feel less present. Having a virtual body provides a grounding for proprioceptive cues, and so if there is a virtual body then the kinaesthetically orientated person is more present and if there is no body they are less present.

The effect of the first perceptual position is also intuitive but the quadratic factor is a problem. It may be that it is an artifact of the measurement process or it could be that some people who generally remember events from the first person standpoint cannot make the suspension of disbelief necessary in order to become present in the virtual environment.

This led to the conclusion that increasing quality of the visual and auditory channels is important, but is not sufficient for presence in the general case. The kinaesthetic sense is just as important and for this reason the virtual body is an essential feature of the system.

3.6 Navigation in Immersive Virtual Environments

As described in Section 3.1.1 the standard IVE system provides tracking over short distance because of the limitations of the tracking range and the tethering of the participant to the virtual environment generator. Thus movement over long distances must be effected with a navigation metaphor, that must be used when the participant needs to move more than a few feet.

As described in Section 3.1.2 two typical navigation metaphors are:

1. Fly in the direction the hand is pointing by pressing a 3D mouse button.

2. Fly along line of sight by pressing a 3D mouse button.

With the standard Provision 200 system, navigation through the virtual environment was performed by pointing in the direction required and

pressing a button. However there is no velocity control and this results in participants occasionally overshooting or undershooting their target. An experiment [SU93] found that participants did become used to this means of navigation though there were in effect two navigation metaphors at work, that is subjects had both an abstract device (the mouse) for navigation but they could also move their own bodies a short distance. Two comments made were:

> Sometimes [I had] a desperate need to actually walk when virtually walking, there does seem to be a conflict between what the eyes see and the body feels - e.g. my feet appear to be floating but I can feel my feet on the ground

> Trying to separate virtual and physical movement: constantly being aware - my initial response was to make the physical move then forcing myself to use the mouse instead ... The amount of concentration I had to use was something I remember particularly. Moving around with the mouse, forwards and backwards - and with the helmet turning around - it was difficult to reconcile the two ways of moving.

This brings out two problems:

- Sensory dissonance. The contradiction between what the subject experiences in the virtual environment and real environment.
- Mixed Metaphor. There are two separate ways of moving - firstly by pressing the button and secondly by moving physically.

A proposed solution, the "Virtual Treadmill" metaphor, was to let the participant walk but walk on the spot [SSU93]. For short distances within sensor range the participants could actually take steps but for longer distances they should walk on the spot facing in the direction they wish to go. This solution removes the need for a button to be used on the 3D mouse, and leaves the hand free to perform grasping and selection.

3.7 The Virtual Treadmill

Hardware devices that could be used in the implementation of the walking on the spot metaphor, such as treadmills , foot switches or extra foot trackers all have one or more of the following disadvantages:

- Cost of purchasing extra equipment
- Extra computation required to process extra data stream(s).
- Extra burden to the participant of suiting up, or further limitations to possible movements.

The solution proposed was a software solution using neural networks. A neural network was designed and trained that could detect when the participant was walking on the spot by the pattern of movement that the head makes on each step [SSU93]. The data used was the position information returned from the Polhemus Isotrak mounted on top of the helmet.

The neural network was a multi-layered feed forward network [HKP91]. The net had two hidden layers of m and n units and a single output unit. The input to the net consisted of l triples, which were the x, y and z relative displacements of the tracker for the previous l time steps. From experimentation values of $l = 20$, $m = 10$ and $n = 5$ gave the best compromise between accurate training and generalization.

There are two type of error that the neural net produces: when the net indicates the participant is walking when they are not (type I), and when the net indicates the participant is doing something else, when they are walking (type II). The first type of error is the worst as it is irreversible, and can be inconvenient when, for example, it causes a participant standing close to an object to collide with it. To overcome this problem we take the neural net output and only change from walking to not walking when a sequence of p 1s is observed and change from not walking to walking when a sequence of q 1s is observed. Again we found from experimentation that values of $p = 2$ and $q = 4$ gave most accurate results for both type I and II

The overall success rate is about 95% and a considerable proportion of the error is caused by the combination of the lag of the Polhemus device and the smoothing of the net output.

3.8 Evaluation of the Virtual Treadmill

The evaluation of the Virtual Treadmill was threefold [SUS94b, SUS95]:

1. To assess the ability of the training methods and net used to learn various walking behaviours.
2. The assess whether the walking metaphor is preferred over the pointing metaphor.

3. To assess the affect of the metaphor on the subjects' sense of presence.

To accomplish this two case control experiments were designed in which two groups of subjects would have to accomplish a simple task. In each experiment the control group navigated using a point in the direction of flying metaphor first and the experimental group used the virtual treadmill first. All subjects saw a virtual body representation (§3.4).

3.8.1 Performance of the Virtual Treadmill

The first experiment was designed to evaluate the training and performance of neural net and assess the participant's preference for navigation method. There was a simple task for each subject to perform after a brief training period to get used to the navigation metaphor. The environment consisted of a corridor with a doorway at the far end leading to a room with a chair suspended out of reach over a deep pit. The task was to pick up a small cube at the dead-end of the corridor then go into room at the other end of the corridor, and to place the cube on the chair.

16 subjects took part and training data was gathered and neural nets trained for all subjects regardless of group, with training taking about 30 minutes per net on a Sun Sparcstation 2. The training data was obtained on the day prior to the actual experience with the participant miming actions such as walking on the spot, picking objects off the floor, turning and looking around the room in a featureless virtual environment for 5 minutes. This data was partitioned into 2 sets, one set to train the nets and one to evaluate the performance.

The performances for the individual neural nets are shown in Table 3.1. Mean success rate on the evaluation data was 91%, with the Type I error being 10% and the Type II error 6%.

Given the subjects unfamiliarity with virtual environments, and the lack of time available to train the neural nets these results were very promising. This is especially true since during the data gathering they were simply instructed to walk on the spot without any motion through the environment taking place.

3.8.2 Ease of Use of the Virtual Treadmill

Two people had to be dropped from the experiment, one simply because they withdrew after the training period and one was removed because they were unable to use the "point and fly" metaphor, an anomaly amongst the several hundred who have used the system.

Subject ID	*Success Rate* %	*Type 1 error* %	*Type 2 error* %
1	92	11	2
2	93	11	3
3	91	15	3
4	85	10	12
5	85	11	5
6	85	15	16
7	92	11	4
8	89	12	9
9	84	6	5
10	90	10	8
11	93	9	5
12	92	7	10
13	92	12	4
14	92	11	5
15	96	6	2
16	95	6	4

Table 3.1: Virtual Treadmill Performance

A questionnaire given to the subjects after the first experience contained three questions designed to ascertain how easy it was to use the metaphors, see Table 3.2. In summary the subjects found using the neural net a little easier in general, more straightforward to get from place to place, and maybe slightly more natural, though all these conclusions are tentative given the small sample used.

Comparing an individual's answers to the three questions against the performance of the neural net they used was also interesting since it indicated that better performance of the neural net leads to more high answers to these three questions. Figures 3.7(a,b,c) show a general correlation between low Type I error and higher responses to the three questions.

3.8.3 Effect of the Virtual Treadmill on Presence

The second experiment was to evaluate the effect of the Virtual Treadmill on presence. The experiment was similar in design except that in the final room reaching the chair was accomplished by either walking straight across the pit, or by a long detour around the pit on a wide ledge.

Again 16 subjects took part, split evenly between the control and experimental groups.

Subjective presence was reported by three questions in the questionnaire, reproduced in Table 3.3. These scores were combined by counting 6 or 7 responses from the three questions giving a value between 0 and 3.

A second, non-subjective measure of presence was whether or not the participant crossed the void or walked around the side

The analysis showed that for the experimental group, the higher the association with the virtual body the higher the presence, whereas for the control group there was no correlation. Secondly those who walked across the void reported lower presence score which might be expected since they hadn't accepted the "reality" of their being a void there.

3.9 Extensions to Virtual Treadmill Metaphor

Prompted by the consideration of a fire-fighter training application, the Virtual Treadmill metaphor was extended to ascending and descending stairs and ladders [SUS94b].

The Virtual Treadmill on it's own applies to constrained motion on the horizontal ground plane. However in certain spaces we would like to be able to move vertically through the space without resorting to "hand flying".

General Movement	**Getting From Place to Place**
Did you find it relatively "simple" or relatively "complicated" to move through the computer generated world?	How difficult or straightforward was it for you to get from place to place?
To move through the world was ...	*To get from place to place was ...*
1. very complicated ... 7. very simple	1. very difficult ... 7. very straightforward
Mean Response	
Control Group: 5.0 n = 6	Control Group: 4.9 n = 6
Exp. Group: 5.1 n = 8	Exp. Group: 5.5 n = 8

Natural/Unnatural
The act of moving from place to place in the computer generated world can seem to be relatively "natural" or relatively "unnatural". Please rate your experience of this.
The act of moving from place to place seemed to me to be performed ...
1. very unnaturally ... 7. very naturally
Mean Response
Control Group: 3.4 n = 6
Exp. Group: 3.9 n = 8

Table 3.2: Ease of navigation questions

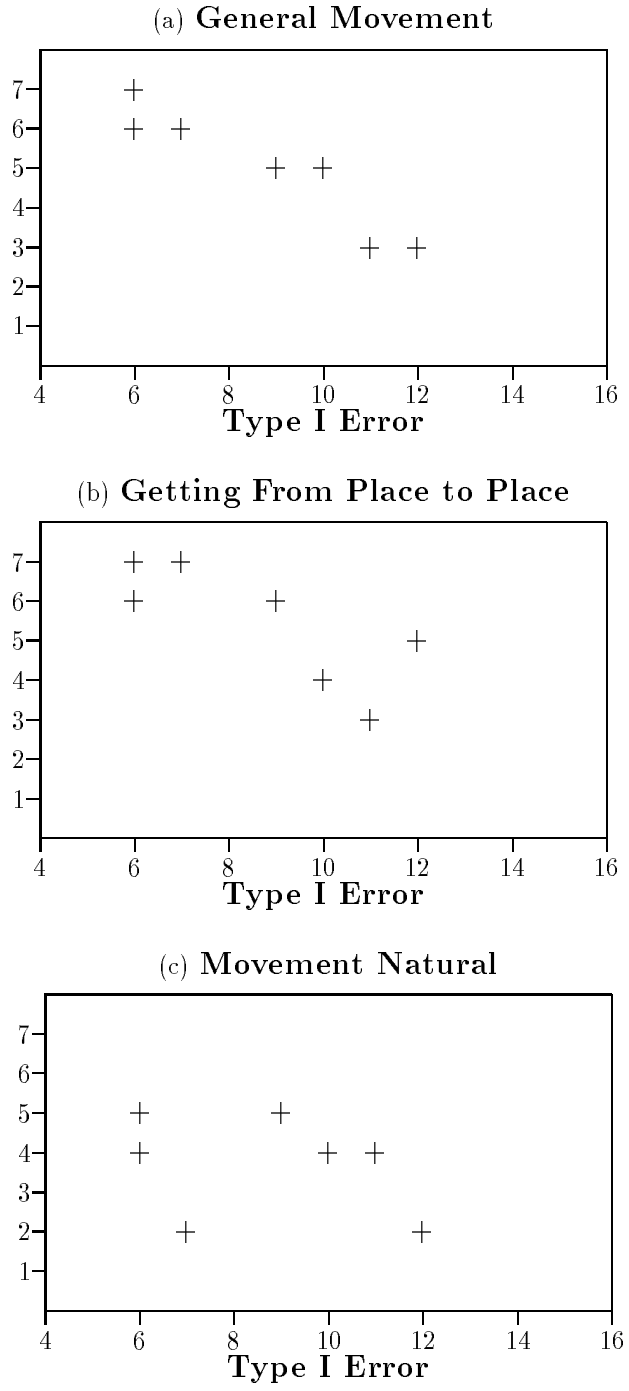


Figure 3.7: Evaluation of navigation by type I error

Please rate your *sense of being there* in the computer generated world on the following scale from 1 to 7:

In the computer generated world I had a sense of "being there"	Please tick against your answer
1. not at all	1
...	...
7. very much	7

To what extent were there times during the experience when the computer generated world became the "reality" for you, and you almost forgot about the "real world" outside?

There were times during the experience when the computer generated world became more real or present for me compared to the "real world"	Please tick against your answer
1. at no time	1
...	...
7. almost all of the time	7

When you think back about your experience, do you think of the computer generated world as more *something that you saw* or move as *somewhere you visited*?

The computer generated world seem to me to be more like	Please tick against your answer
1. something that I saw	1
...	...
7. somewhere that I visited	7

Table 3.3: Subjective presence questions

The climbing of steps and ladders can be achieved by a simple extension to the Virtual Treadmill. The requisite extension to detect collision between the participant and the model of the steps or ladder inside the virtual environment. For a staircase we can move the participant up and down to remain on the individual steps depending on their direction of motion. However for vertical ladders, the participant can move up or down at any point, so to distinguish which action was desired the participant either held their hand above their head for upwards and below the waist for downwards. This gesture suggests but does not actually mimic the action of climbing a ladder. At any time, if the participant moved so that the legs of their virtual body were off the step or rung they are on, they "fall" to the ground if this were actually possible in the environment.

As simple usability study with five participants showed that this metaphor was useful and they all completed a given task within a reasonable time of 8 minutes where an expert had taken 3 minutes.

3.10 Presence Model

The results discussed so far in this chapter suggest a more general model of how a particular participant might react to virtual environment. This is that presence is a function of two main factors:

- Match between internal representation and sensory data.
- Match between proprioception and sensory data.

The first factor is the match between the sensory data and the representation system that the participant uses. The match is required so that the participant can form an internal model of the environment. Thus as discussed in Section 3.5, a vivid visual display might afford a visually dominate participant a high level of presence, but another, who relies on sound to form their world model a low sense of presence.

The second factor is the match between proprioception, the sense of what state one's body is in and the sensory data. An example of this is the Virtual Treadmill where the match between the optical flow experienced during motion through the environment, and the nearly match proprioceptive state of walking on the spot. In general the changes to the sensory data over time must match what the participant expects to happen when they make an action due to their knowledge of proprioceptive causal relationships in the past. This does not exclude fantastic virtual environments, but to become present in them the participant must become accustomed to the rules of cause

and effect in the environment. A corollary of this is that when designing interaction metaphors and gestures, they should relate as close as possible to the task that will be performed, a design paradigm called *Body-Centred Interaction* [SU94a].

The factors that affect presence that were listed in Section 3.3 can be used to elaborate this model, see Table 3.4[SUS94a].

Vividness	High quality information presented in an unobtrusive manner
Consistency	Sensory data should be consistent.
Interaction	There should be many possible interaction within the environment
Virtual Body	Self-representation should be anthropomorphic and correlate to the movements of the participant
Cause and Effect	The participant should be able to model the cause and effect relationships within the VE

Table 3.4: Factors affecting presence

Vividness [Ste92] and **Consistency** concern the degree of immersion that the display gives and provide a grounding upon which the match between sensory data and internal representation can take place. The **Virtual Body** affects both the match between sensory data and internal representation. Firstly the participant should see a body in the virtual environment, since they do so in reality and is part of their internal representation of themselves in an environment. And secondly the virtual body is one of the cues the participant will rely on to gauge the effect of actions they make, and so should mirror what the participant's proprioception tells them about the position of their body. Other rules of **Cause and Effect** should also maintain this match between sensory data and proprioception. **Interaction** should of course maintain the match between sensory data and proprioception, but a range of interactions also allows the participant to become engaged in tasks and model the **Cause and Effect** loop of the environment, through performing those tasks.

3.10.1 Further Results

Further confirmation of this model and further results on factors affecting presence were provided by another study [SUS94a]. This confirmed the results of Section 3.5.1, presence was positively correlated with visual dominance, negatively with auditory dominance and positively with kinaesthetic dominance since a virtual body was used. Another factor that was found to influence presence was the stacking depth of the environment. The stacking depth referred to the manner in which the participants moved between different rooms within the virtual environment. One group would navigate through doors, whilst the other group would don a virtual head mounted display, which would transport them to another environment. This navigation method was meant to mimic the way in which the participant enters the first virtual environment by putting on a real head mounted display. Presence was found to be positively correlated with depth, i.e. the number of environment visited, when using the virtual HMD and negatively when having to move through the doors.

In another study the addition of 3D sound cues was tentatively found to switch the correlation of auditory dominance with presence from negative to positive [Pat94]. This was as expected since as noted in Section 3.5.1 the Provision 200 system used for the initial experiments had poor support for sound effects, whereas the system used for the 3D sound study was a later model, a Provision 100 VPX with Beachtron sound card and Akai 3200 sound sampler.

3.11 Conclusions

This chapter has shown how interactions metaphors combined with the representation within the environment can provide an enhanced sense of presence within a virtual environment. In turn such a sense of presence can lead to a participant's behaving naturally within the virtual environment and using known skills to perform tasks. The effect of interaction and presence is circular in that certain interaction metaphors allow the participant to match sensory data with proprioception allowing them to perform tasks within the environment which in turn reinforces the sense of presence.

The presence model arrived at in Section 3.10 indicates several criteria to aim for when constructing and using interaction techniques in virtual environments. The *body centred interaction* paradigm maintains that making interaction techniques mimic the task in reality should allow the participant to not only comprehend the metaphor more clearly but be able to model the

causal rules of the environment easily.

A further example of this paradigm is world scaling metaphors [SU94a]. Rather than using iconic hand gestures to scale the world [BHV92], the gesture used is that of scaling oneself. To scale the world up, you make the gesture of squashing yourself, by pushing down on your head and crouching down. And to scale the world up you crouch and pull yourself up by mimicing grabbing your head and standing.

However experienced participants may want to be able to use iconic gestures as shortcuts much as experienced window environment users use keyboard short cuts. Given the subtlety of expression of human gesture, such an interface could be very broad, given accurate tracking on a number of sensors using the techniques described in Sections 3.7 and 3.2.2. Creating such gestures would most easily be accomplished within the virtual environment as then the participant can demonstrate the gesture in context and with reference to their virtual body representation within the environment. Also a gesture such as tapping one's belt to bring up a virtual tool belt is a participant-specific gesture depending on body shape and would need to be recognized slightly differently for each participant.

The idea of creating interactions by demonstration by referring to one's representation within the virtual environment is one of the motivations for the VEDA system described in Chapter 5.

Chapter 4

Programming Interactions with Virtual Environments

With many applications now being developed for immersive virtual environment systems there is a growing need for effective tools to create and manipulate the environments that will be presented. To date most authoring systems have focussed mainly on the appearance of the environment that is, the geometry, colour, lighting and position of objects within a three dimensional scene. This is epitomized by the recent generation of VRML 1.0 (Virtual Reality Modelling Language) [BPP95] browsing and authoring tools which concentrate on the presentation, but not the behaviour and interaction with objects.

The behaviour of objects and especially the interactions of the user with the environment are usually hard-coded within the application that generates the virtual environment. Various toolkits have been designed to assist within the construction of these applications and some current systems allow some scripting of simple objects behaviours and provide a standard application that will display these.

Still, the usual method for programming most systems is to display the environment, enter the system and view it by putting on a helmet or shutter glasses, then come out make changes and either re-run or at worst re-compile the application.

To escape this loop methods can be borrowed from the field of visual programming languages and the eventual aim of the next chapter is to show how an immersive virtual environment can be programmed from within, that is whilst the participant is immersed, without leaving the environment and without having to re-run or re-compile the main application.

This chapter reviews current programming systems for virtual environments and briefly reviews the areas of visual programming most relevant for

the eventual construction of an immersive programming language for virtual environments as described in Chapter 5.

Section 4.1 details various scene description languages in order to illustrate the fundamentals of virtual environment design. Section 4.2 illustrates the breadth of programming paradigms for virtual environment libraries and the services that they provide. Section 4.3 gives an overview of visual programming languages. Section 4.4 gives specific detail about data flow visual languages since this will form the basis of the paradigm used in Chapter 5. Sections 4.5 and 4.6 review non-immersive and immersive visual languages for programming virtual environments

4.1 Virtual Environment Scene Description

Static scene description languages are typified by Virtual Reality Modelling Language (VRML)1.0 [BPP95, Har95]. VRML is a hierarchical scene graph language, where each node in the graph can have child nodes that inherent all the current properties, such as transformation, texture and colour, of their parents. To render such a graph it is traversed in order, with properties accumulating until a geometry node is reached which is rendered with those current properties. There is a structural mechanism that allows the scope of such properties to be limited and since VRML was designed for integration with the World Wide Web and its browsers, VRML scenes can contain hyperlinks to other scenes or other document types.

The basic nodes of the VRML language are:

- Geometry
- Appearance
- Lights and Cameras
- Transformation
- Structure

Geometry nodes include basic building blocks as Cube, Cylinder, Sphere and more general geometry sets: PointSets, IndexedLineSets and IndexedFaceSets. The set nodes are defined with reference to an array of 3D coordinates specified by a Coordinate3 node. A PointSet node contains a range of indices which reference the array of coordinates specified by the Coordinate3 node. IndexedLineSet contains one or more lists of indices into the coordinate array which specify the line's path. IndexedFaceSet contains one or

more lists of indices into the coordinate array which specify a closed polygon. It is also possible to specify normals for each vertex, line, face or object using the Normal and NormalBinding nodes[1]. Since IndexedFaceSets can define arbitrary shapes, another node, ShapeHints, can be used to give the renderer directions about how to optimize drawing because of the nature of the object being defined.

Appearance nodes specify materials and textures to apply to lines and faces. Materials can be specified for each vertex, line, face or object by using the Material and MaterialBinding nodes. The Material definition contains information about ambient colour, diffuse colour, specular colour, emmisive colour, shininess and transparency [2]. Textures are more complicated. Firstly a Texture2 node defines a file to use as the texture and whether the textures tiles the two possible directions. Secondly a Texture2Transform node specifies how texture coordinates are applied over the object. Finally a TextureCoordinate2 node contains an array of 2D coordinates that can be indexed when defining line and faces that will give the 2D texture coordinate of each 3D vertex.

Lights and Camera Point and directional lights are supported. Both lights have a colour, intensity and flag to determine whether or not they are on. Two types of camera are supported, Perspective and Orthographic.

Transformation Each object is defined within a local coordinate system, which is transformed by a modelling transform into world coordinates. Transformations provided are Scale, Orientation, Translation, and Transform. Transform is a combination of a scale, orientation and rotation, along with a centre point about which the individual transformation take place. Transformations are relative to the current modelling transform defined in the scene graph so objects that are children of other nodes inherit their transformation and concatenate their own to arrive at their modelling transform.

Structure Separator nodes provide the mechanism with which properties are given scope. When a Separator node is encountered the current state

[1] In fact the Normal node contains an array of vectors to use as normals, and the NormalBinding describes how to use this array when defining subsequent geometry. Binding types include to specify normals for whole objects, for part objects, for each vertex or for each face. The normal can also be used in order from the array or via an index specified as part of the IndexedLineSet or IndexFaceSet lists.

[2] The bindings for materials are similar to the bindings for normals.

is saved (state includes cameras, lights, coordinates, materials etc.), and reloaded once the child nodes have been traversed. A rudimentary form of object culling can be performed by using separators where each separator maintains a bounding box of its' sub parts that can be culled against the view volume. WWWInLine nodes allow VRML objects to encapsulated and incorporated into other scenes. It specifies a VRML file via an arbitrary URL which is inserted into the scene graph at the current point. WWWAnchor nodes specify hyperlinks to other scenes or other types of documents. A final interesting structural mechanism that VRML provides is level of detail switching with a LevelOfDetail node. This node contains several values that correspond to screen areas. If the screen area of the rendered geometry is greater than the first value then the first child node is rendered in the next frame. If it is between the first and second values the second child is drawn and so on.

Figure 4.1 shows the description of an example scene containing two objects. It consists of a single camera at the origin and a slightly green light that is behind and to the right of the camera when the scene is entered. There are two separators following the camera definition each containing a complete object description. The first is a yellow cylinder in front and to the left of the initial camera position. The cylinder itself is specified by it's radius and height, and the current properties are rotation by $\pi/4$ radians around the X axis, translation by the vector $(-3, 0, -10)$ and a material which is yellow. The material binding isn't set and so defaults to the material being applied to the whole object. The second object is a cube but an IndexedFaceSet node has been used rather than a Cube node. The Coordinate3 node specifies the eight corner vertices and the IndexedFaceSet specifies the faces as lists of indices into the Coordindate3 node separated by -1. The ShapeHint node here provides a useful optimization for browsers to use. Because an IndexedFaceSet can define arbitrary shapes, it can't be assumed what order the faces are defined in, and therefore no back face clipping can be done. The ShapeHint here tells the renderer that the shape is solid, that is the external surface is complete, so there is no need to draw back facing polygons. The ShapeHint also tells the renderer which orientation the faces are in so it can work out a normal and use that for back face elimination.

The scene as viewed in Webspace from Silicon Graphics [Sil] is shown in Figure 4.2. Webspace is a desktop application that works as a helper application to Web browsers. The controls at the bottom are one of two interaction metaphors used to interact with the scene. The handle in the middle turns the view left and right and moves it forwards and backwards in

```
#VRML V1.0 ascii
Separator {
  DirectionalLight {
    direction -1 0 -1
    intensity 1
    color 0.7 1.0 0.7
  }
  PerspectiveCamera {
    position    0.0 0.0 0.0
  }
  Separator {   # The yellow cylinder
    Material {
      ambientColor 0.3 0.3 0   # Pale Yellow
      diffuseColor 1 1 0       # Yellow
    }
    Translation { translation -3 0 -10 }
    Rotation { rotation 1 0 0 0.785 }
    Cylinder {  radius 2.0 height 2.0 }
  }
  Separator {  # The blue cube
    Material { diffuseColor 0 0 1 } # Blue
    Translation { translation 3 0 -10 }
    Scale { scaleFactor  2 1 2 }
    ShapeHints { vertexOrdering  COUNTERCLOCKWISE shapeType SOLID }
    Coordinate3 {
      point [-0.870000  0.940000  0.732500,
             -0.870000 -0.560000  0.732500,
              0.560000 -0.560000  0.732500,
              0.560000  0.940000  0.732500,
             -0.870000  0.940000 -0.732500,
             -0.870000 -0.560000 -0.732500,
              0.560000 -0.560000 -0.732500,
              0.560000  0.940000 -0.732500]
    }
    IndexedFaceSet {
     coordIndex [0, 1, 2, 3, -1,
                 6, 7, 3, 2, -1,
                 7, 6, 5, 4, -1,
                 4, 5, 1, 0, -1,
                 1, 5, 6, 2, -1,
                 4, 0, 3, 7, -1]
    }
  }
}
```

Figure 4.1: An example VRML file

the direction of view. There is a small dial on the right of the handle that corresponds to looking up and looking down. The cross hairs on the left can be used to specify a point on an object to approach. The right hand control slides the view up and down and left and right in the direction of the view window.

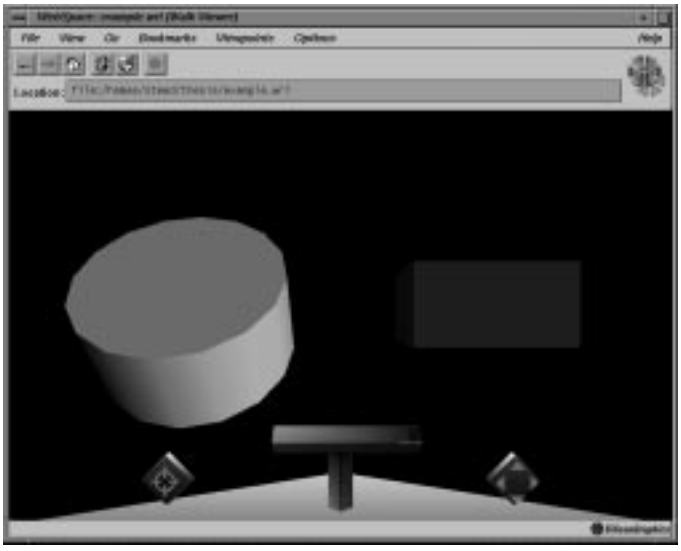


Figure 4.2: Example VRML scene viewed in Webspace

Other scene description languages provide much the same visual elements, though some deal with more general constructive geometry and some only allow object descriptions with simple triangles since they are optimized for certain processors. However within VRML there is no ability to specify sounds, behaviour, interaction or networking.

Sounds can easily be added to a scene and simple behaviours can be described in terms of reactions to known events such as Collision, Selecting and Picking. The MAZ file format from Division [Div94a], an example of which is shown in Figure 4.3, provides both such abilities.

This example defines a toy plane that follows a spline path once the event SCENE_ENABLE has been received that indicates the start of the animation in the virtual environment and will respond to being touched by the participant by playing a sound. It also has a propeller object as a child which rotates in its local coordinate system whilst attached to the plane and moving in the plane's coordinate system as well.

Some scene description languages have been extended until they become complete programming languages where arbitrary behaviour can be described. The DIVE system has a language that allows object to contain

```
OBJECT splane {
    VISUAL {
        GEOMETRY "office/splane"
    }
    EVENT {
        SCENE_ANIMATE:
            followSpline(*, "office.spl", 0.01, -1);
            BREAK;
        TOUCH:
            soundOn (radiototower, -1);
            BREAK;

        UNTOUCH:
            soundOff (radiototower);
            BREAK;
    }
    OBJECT propeller {
        VISUAL {
            GEOMETRY "office/sprop"
        }
        EVENT {
            SCENE_ANIMATE:
                spin(*, 0, 10);
                BREAK;
        }
    }
}
```

Figure 4.3: A MAZ file example

```
object {
    material "red"
    view {
        SPHERE
    }
    begin.tcl
        proc move_up {type src_id id pid} {
            dive_move $id 0 0.5 0 LOCAL_C
        }

        dive_register INTERACTION_SIGNAL DIVE_IA_SELECT move_up
    end.tcl
}
```

Figure 4.4: DIVE Tcl extension example

tcl scripts [Ous94] to be executed as events arrive [CH93, FH95], blurring the distinction between the scene description and the virtual environment generator or browser providing the underlying global services for the virtual environment such as collision detection and device interaction. Figure 4.4 shows an example of a script that defines a red sphere that moves up when the user selects it.

4.2 Virtual Environment Programming Libraries

Programming systems for virtual environments cross the full range of programming paradigms, from low-level imperative programming libraries, to object-orientated toolkits, to declarative systems for environment description, to interpreted scripting languages.

The most basic form of virtual environment programming library is the graphics toolkit for interactive rendering such as OpenGL, a standard designed by Silicon Graphics but adopted by many workstations vendors, or Renderware designed for fast software rendering on PC platforms. Other libraries are usually built around such a rendering library but with services to aid in the specification of virtual worlds, though for applications requiring optimal performance a last resort is to return to the root rendering system and ignore the overhead of the VE toolkit. The virtual environment programming libraries might include extra services such as functions to load

object geometry, interact with peripheral devices, transform objects, and detect collisions between the geometry of objects.

Many toolkits now exist, the most widespread of which is WorldToolKit from Sense8 Corporation. It provides a C library that covers all aspects of virtual world application building such as:

- Sensor configuration and interaction
- Loading of various object file formats and dynamic geometry
- Rendering
- Collision detection
- Object hierarchies
- Event Handling

The WorldToolKit API contains over 650 function calls, but scene description is limited in the base API to the hard coding of the behaviours of objects in C leading to a compile and test cycle as the application is refined.

A WorldToolKit application runs with a simulation loop that executes in serial order:

1. Read sensors
2. Call actions functions for whole world and individual objects and update objects with sensor data
3. Render universe

However as noted in [RCM89] a simple simulation loop leads to two problems: *multiple agent problem* and *animation problem.*

The multiple agent problem occurs when the various simulation and input/output device agents, which will have different time constraints, are competing for processor time. A good example of this problem is given by systems that use a Dataglove for input. One way of getting data from the glove is by 'polling' the glove, where the device interface requests data from the glove and then has to wait until the data arrives before it can carry on. Such a system will be seriously slowed down by the lag rates incurred by the use of the magnetic tracker.

The animation problem then arises when the system is supporting smooth animated graphics and also the various agents described above. The specific problem here is that we wish to have the best possible but constant frame

rate for the display, but however we don't want to achieve this by slowing the system down so that no matter what demands the applications and devices make, the frame rate will not decrease.

One solution is to distribute the various processes over several processors, with the graphics rendering being the most obvious choice because of it's high processor demands and the availability of dedicated acceleration boards. In fact one of the early dedicated virtual environment systems the RB2 system [BBH+90] from VPL was designed in precisely this manner with a Machintosh controlling the virtual world and input/output devices and two Silicon Graphics Irises rendering the graphics.

This paradigm of separating the virtual world into many processes running concurrently was the basis for the Minimal Reality Toolkit from the University of Alberta [SLGS92]. Their *Decoupled Simulation Model* broke virtual environment generation up into four distinct components, Computation, Presentation, Interaction and Geometric Model. Figure 4.5, adapted from [SLGS92], shows the relationship of these components and a fifth autonomous process - the participant. The original paper's parenthesised elements indicate unimplemented elements.

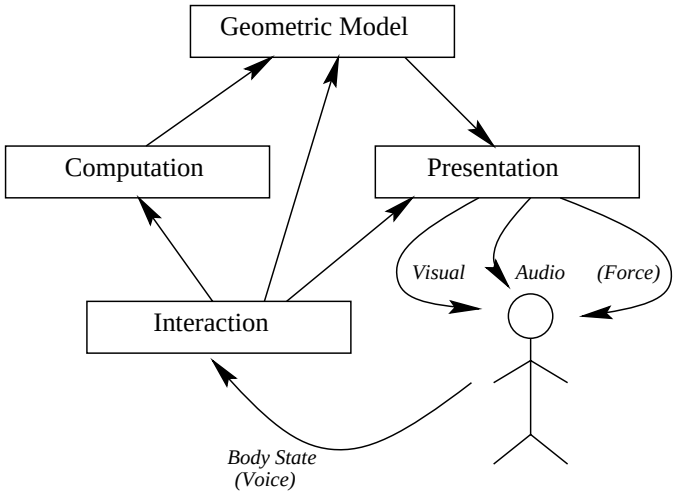


Figure 4.5: Elements of the decoupled simulation model

The Interaction component handles device driving and receives body state including positions and gesture information.

The Computation component manages all non-graphical elements of the application and runs a simulation model which is driven by the participant's interaction with the environment.

The Geometric Model maintains the database of the environment. It is driven by the Computation element, since it converts relevant data from

the computational model into visual, audio and force data. It is also driven by the Interaction since the actions of the participant should be presented immediately whilst their effects are processed by the computation element. An example of this would be presenting visual feedback of a participant's arm moving as soon as they move it, and before collision detection can be carried out, which might then determine that they tried to move it through a solid object.

The Presentation component then renders the Geometry with respect to the position that the participation is currently in.

An application thus consists of several concurrent processes that can be run on separate processors. These processes perform different tasks and communicate common information via a data sharing package which supports one way communication. The MR toolkit provides several library packages that aid the construction of these processes. Apart from the data sharing package, there are packages for different devices, packages for visual and sound displays a Workspace Mapping package that converts device coordinates into environment coordinates, a Panel packages that provides a way of implementing 2D style interfaces in a 3D environment and a Peer package that allows two MR toolkits application processes to commmunicate messages between the group. At this level it provides similar services to WorldToolKit.

Many other toolkits exist though they vary in the exact devices and object formats they support, the ability to support multiple participant and multiple worlds and the ability to distribute tasks amongst available resources. Division's dVS environment and VC library [Div94b] will provide a more in-depth example since the work described in the next chapter will be based upon these.

4.2.1 dVS and the VC Library

The VC library is a high-level toolkit for creating virtual environments that relies upon dVS, a runtime environment that provides services essential for creating virtual envrionments. An application written in the VC library does not need to manage the processes that generate the virtual environment displays, but works at an object database level, where it manipulates and accesses a world database that describes objects, their properties and events within the environment. dVS runtime provides several processes, or actors, that also access this database and act upon it. These include:

- *Visual actor* renders graphical views of the object database.
- *Audio actor* renders auditory views of the object database.

- *3D tracking actor* manages tracking devices and processes their raw data.
- *Collision actor* processes object movements and generates collide events.
- *Body actor* generates an object that represents the participant within the environment.

The relationship between these software layers is shown in Figure 4.6, adapted from [Div94b]

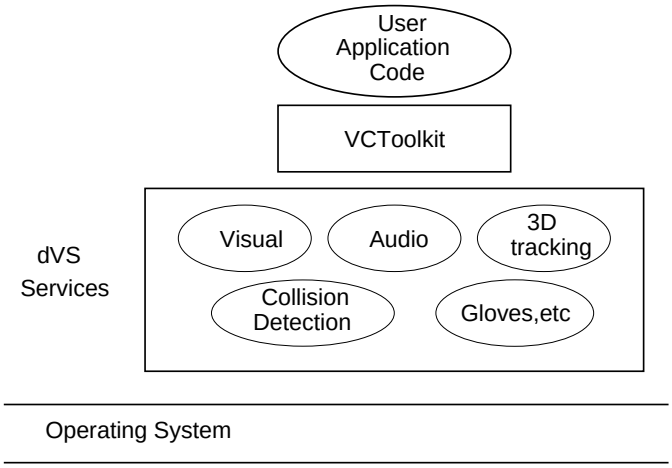


Figure 4.6: Relationship of dVS and the VC library

The initialisation of a VC application involves adding objects to a database, and registering a set of callbacks that depend on events generated by the other actors. Each actor generates events that reflect the services it provides. The body actor returns the position of the head and the hand, and the state of any button device the user is holding. If there is a glove, then a glove actor returns the posture of the hand and so on.

The object database contains properties of objects similar to those that scene description languages provide (§4.1). Indeed the MAZ file format described in Section 4.1 is designed is be loaded and animated by a standard application written in the VC library. In a MAZ file, events trigger VC coded functions from a standard set, see Figure 4.3, but more complicated behaviours must be coded by writing VC code directly. Figure 4.7 gives the outline of some example code that creates an environment with two spinning cubes.

The VC toolkit thus provides a general programming system for virtual environments. The actor model is well-suited to distributed parallel processing. Collaborative envrionments are also simple since more than one set of

```
#include <vctools.h>

VCEntity_Ptr object1, object2;

/* Callback function to rotate the objects, this gets called
   each time the interrupt fires */
void alarmHandler(void *data) {
  VCSpin(object1,VC_Y,20.0);
  VCSpin(object2,VC_Y,20.0);
}

main(int argc, char *argv[]) {
  ActorId     actor_id;
  VCEntity_Ptr eptr;
  Object_data objects;

  /* Initialise an application Actor and collect process args */
  actor_id=VCApplicationInit(&argc, argv);

  /* Create a light */
  eptr = VCConstruct(NULL,NULL);
  VCLighting(eptr, VC_AMBIENT_LIGHT, NULL, NULL);

  /* Construct two objects */
  objects1=VCConstruct(NULL,NULL);
  objects2=VCConstruct(NULL,NULL);

  /* Populate the objects with geometry */
  VCVisualise(object1,"misc/teapot",NULL);
  VCVisualise(object2,"misc/teapot",NULL);

  /* Shift one of the objects away from the origin */
  VCTranslate(object2,15,20,0);

  /* Set up all the call backs to allow picking and moving  */
  VCAddSignalCallback(SIGALRM,alarmHandler,(void *) data);

  /* Start the main event handling loop */
  VCMainLoop();
}
```

Figure 4.7: VC code example

rendering and device processing actors can be attatched to the central object database provided by the dVS runtime.

4.3 Visual Programming Languages

The term "Visual Programming" has come to include a whole collection of techniques and environments that use visual constructs to build programs or display output [Cha87, Shu86, Mye86, Shu88, aMMB89, Cha90]. Shu's taxonomy breaks the subject up under the following headings [Shu88, Chapter 1]:

- Visual Environments
  - Visualization of data, program execution or software design
  - Visual coaching
- Visual Languages
  - for handling visual information
  - for supporting visual interactions
  - for actually programming

The area most relevant to the research described in the next chapter is diagrammatic data flow programming languages, but because the environment within which you program is also the one in which the programmed actions take place, there are elements of visualization of data objects and program execution, since the environment itself contains the program and the program manipulates the environment. There is also an element of visual coaching in that, for example, training a gesture recognizer is done by demonstrating the action to be recognized and certain behaviours of objects can be demonstrated.

4.3.1 Visualization of data, program execution or software design

These systems provide graphical views of the entities and characteristics of a computer system or program [SP92a, PBS93]. Statsko *et al.* identify a four-dimensional characterization of these visualization systems:

- Aspect

- Abstractness
- Animation
- Automation

The different **aspects** of the system that can be visualized are the program text, data structures, run-time state, control flow or algorithmic methods.

Abstractness relates to the type of visualization system used. For example an integer might be represented as a counter if its range is unknown, but at a higher level of abstraction, where it is known to be a percentage, it might be depicted as a pie-graph.

Animation refers to whether the visualization is dynamic or not. For an example a flowchart visualization of a program might be animated so that the statement being executed was highlighted.

The level of **Automation** describes the way in which the visualization is formed, either by interpretation of the memory at run time or by the addition of explicit visualization commands at run-time.

Perhaps the oldest form of visualization system is the data flow diagram or pretty printer for program code [Tri88]. These systems display the code in a graphic structure that highlights certain aspects of the code's structure such as branching, recursion or looping.

The best examples of data visualization systems is the Balsa family [BS84, BS85, Bro88b, Bro88a]. Balsa is an animation system initially used to teach aspects of algorithms to students. The animation of, say, a depth search algorithm traversing a graph is created by the augmentation of the program code with visualization prompts. Thus the degree of automation is low but the resulting animations can be quite abstract and informative.

Other examples of visualization systems include those from Brown University [Rei85, Rei87, RGR89], PROVIDE [Moh88], PVS[Fol86], Animus [Dui88] and PegaSys [MH85]

4.3.2 Visual coaching

These systems attempt to provide some programming support by inferring some of the required code by monitoring the user working through the problem or supplying input-output pairs. The emphasis is on demonstrating in some way constraints to apply or behaviours for objects to adopt without resorting to 'telling' the objects what to do with explicit code.

Pygmalion integrated this ability with one of the first visual programming languages [Smi77]. The basis of this system in an editor that can remember

the sequence of operators applied to a set of icons. Smith distinguishes his system from other visual programming systems so:

> PYGMALION has no representation for *telling* a program anything; PYGMALION is an environment for *doing* computations. If the system happens to remember what is done, then a program is constructed as a side effect

Thinglab is a simulation system that uses constraints to specify the relations between the objects of the simulation [Bor81]. It supports graphical specification of these constraints by giving two views of the object *use* and*construction*. The use view gives the appearance of the object whereas the construction view shows the composite objects and the constraints between them. The constraints can be created by demonstration. For example making a point lie on a line can be demonstrated by placing the point over the line. [Bor86a, Bor86b].

With the Peridot system users can create user interface management systems by creating an interface and specifying the various widget parameters by giving example values [Mye87, Mye90a].

The Programming by Rehearsal system's metaphor is that of a stage where *performers* interact by sending *cues* to each other [FG84]. The programming of a system proceeds in five steps:

- Audition various performers by observing their reactions to certain cues.
- Place all the required performers on the stage.
- Block the performance by positioning and resizing the performers.
- Rehearse the production by showing each performer how it should react to the cues sent to it.
- Store the production for later use.

A production then consists of the actions taking place on the stage, possibly supported by other off-stage performers.

The Eager system generates program by trying to spot repetitive patterns in the user's activity [Cyp91]. Built around a Hypercard system, Eager can anticipate tasks by highlighting what it expects the user to do next. Once Eager demonstrates that it follows the procedure being demonstrated it can be instructed to complete it.

4.3.3 Visual Languages for handling visual information and visual interactions

These are not considered proper visual programming languages by some researchers since they are primarily text based languages to specify or interact with visual information or support visual interactions [Mye90b].

Examples that Shu [Shu88, chapter 6] gives include Pictorial Structured Query Languages for manipulating pictorial and alphanumeric databases [RL84] and the Spatial Display Management Systems mentioned earlier, where views of standard databases are constructed by associating attributes of icons with tuples of the database [Her80].

4.3.4 Visual Languages for actually programming

These systems allow programming with graphical objects. Early systems evolved from basic program visualization systems, where the program structure is illustrated by the layout, by allowing the program to be constructed within such a view by the addition and editing of the graphical symbols. They can be broken down by the type of visual languages employed, *diagrammatic*, *iconic* or *forms* based.

PIGS [PN83], is an example of such a diagrammatic. It uses Nassi-Scheiderman diagrams that can be directly edited and supports interactive debugging by allowing the user to watch the flow of control around the diagram. PIGS is an example of a *diagrammatic* visual language.

Data Flow diagrams are also diagrammatic, but rather than illustrating the flow of control they concentrate on illustrating the flow of data between application objects and many recent systems are based on this model [Hil92]. Examples are described at length in Section 4.4.

The Pict family of objects characterize *iconic* languages [GT84, GMD90]. Within the Pict environment users can program by using a joystick to place icons that can be connected together. Once the program has been constructed and compiled its execution is animated and the flow of control can be observed passing through the layout. Unlike data flow systems the picture produced corresponds directly to an imperative program. VisaVis is the equivalent system for describing functional programs [PTVM92, PVM94].

Form based systems such as Formal [Shu85] capitalize on the users familiarity with traditional forms to enable them to implement complex data manipulation tasks. The basic approach is to specify a form corresponding to the output, give the source of the information and then specify properties that certain cells must satisfy.

4.4 Data Flow Languages

The basic concept of a data flow language is that of a collection of filters that accept a number of input data streams, process them and then output one or more data streams. The whole system is driven by input filters that sample external events and eventually the data streams reach output filters. The obvious graphical representation is that of a graph with nodes representing the filters and arc representing the data connections.

The types of filter and data allowed depend on the application domain the system is intended for. The original HI-VISUAL system was a data flow language for image processing where filters correspond to traditional signal filters [IH87]. An example HI-VISUAL program might have source nodes such as cameras and filter nodes such as edge detect and binarize.

ConMan also has a restricted domain, that of graphical application development [Hae88]. It is restricted to connecting together pre-built modules, for example a 2D curve editor is connected to a 3D sweep and view module. The data being communicated in this example consists of lists of transformations to specify the views and geometry lists to specify the objects.

The Application Visualization System (AVS) supports scientific visualization applications [CFD+89], though it too is also restricted to the final integration of software modules and is not a general programming system. Serius is a similar system for the Machintosh environment and has more emphasis on constructing the user-interface [Lan91].

Systems have been developed that allow more general programming. Show and Tell is designed to introduce children to the concepts of programming [KCM90]. Example programs use various sets of filters, from logic gates and arithmetical symbols to phone dialing and printer output.

Prograph 2.0 is another programming system that integrates aspects of AI languages and the object-orientated paradigm into a data flow language [CP88, Gol91]. A data flow language is used to specify methods of the objects.

Cantata is a data flow language that forms the basis of the Khoros application development system [RW91]. Cantata is multi-paradigm in that the filters are connected is a node and arc type editor, but the filters themselves can have parameters that are specified using forms.

Another area where data flow programming has been successful is in the specification of user interfaces. One example of this is the Fabrik programming environment [LCI+88, IWC+88]. The filters in this case include graphical manipulation filters alongside arithmetic and string handling filters. Data may also include graphical objects and the flow of data may be bi-directional.

4.5 Visual Programming for Virtual Environments

Distinction must first be made between visual languages to *describe* virtual environments and visual languages that are presented *within* virtual environments.

The first type of system are visual languages for describing virtual environments that are used external to the environment on a desktop system. xDVISE is an X-Windows program that generates a desktop display of the object hierarchy during its' running which allows customization of object properties and the hierarchy interactively at run time through a forms interface [Div94c]. Figure 4.8 shows 3 windows, one with the virtual world showing a view of a cooker in a kitchen, one showing the part of the object hierarchy where the cooker is defined and one showing the part dialogue defining the object properties, specifically position, orientation and scale.

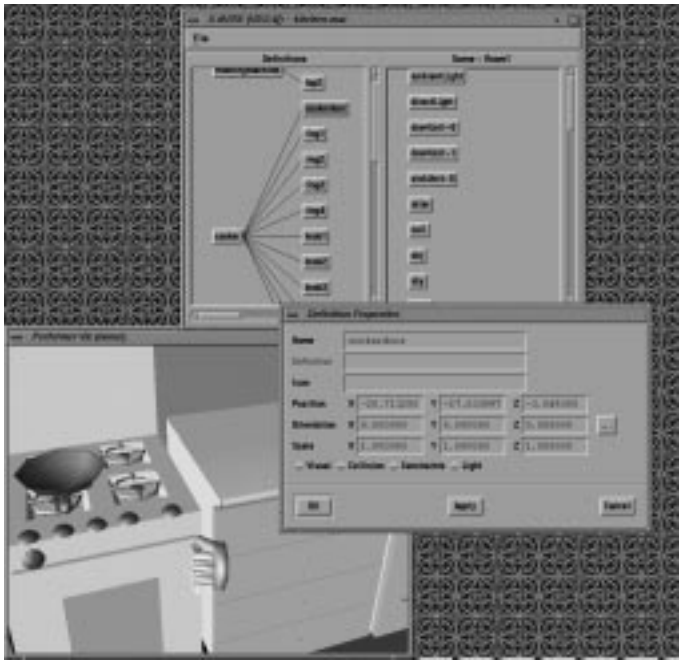


Figure 4.8: xDVISE display of a kitchen showing the scene hierarchy and a property form

A similar desktop display is used in the VR-MOG system which also provides a 2D layout tool as well as a forms based properties editor [CRP95]. In addition it also allows editing of event based behaviours at a level similar

to that provided by the DIVE toolkit scene description language (§4.1), from which it is derived.

Body Electric [Kal93a, pages 212-219] from VPL is a data flow language that was used to describe virtual environments on the RB2 (Reality Built for 2) system [BBH+90] . Data flows from the input devices through data 'massage' units to drive events in the virtual environment.

The AVS system for scientific visualization has also been extended to support visualization within virtual environments [She93].

The second type of system involves a visual language which rather than being manipulated on a 2D display is manipulated within the virtual environment.

Glinert proposes extending his BLOX method which uses flat jigsaw like pieces to 3D using cubes that can be snapped together much like childrens building blocks [Gli87].

Producing 3D animations is the aim of at least two visual programming systems which illustrate different directions these tools can take. Virtuality Builder II [GB95] is an integrated 3D environment in which constraint and object paths can be specified to create animations. Though not a complete programming language, Virtuality Builder II can be used to specify many complex animations. VPLA [Lyt95] is an interpreted language specified by a 3D network editor that describes components of an animation in terms of hierarchical objects and actions on them such as transformations, modelling operations, deformations, particle systems and recursive procedures. VPLA's output is a RIB file for subsequent non real time rendering by Pixar's Renderman.

Three virtual environment languages for actually programming that have been implemented are CUBE [NK91, NK92, Naj94], Lingua Graphica [SP92b] and CAEL-3D [vRCBF95]. CUBE uses a 3D data flow metaphor to describe logic programs. Two versions of CUBE are reported. The first CUBE-I simply allows visualization of the result of evaluating a prefabricated program, whilst CUBE-II, illustrated in Figure 4.9, is an interactive desktop editor.

Lingua Graphica is a 3D editor for a C++ base language. The Lingua Graphica workspace looks like a tool board with the tools being the various library functions, primitives and pre-defined types. These can be juxtaposed with the relations between the objects corresponding to the syntax rules of the language, see Figure 4.10. The programs Lingua Graphica was designed to modify were in fact the virtual environment description, but when the programmer made changes within the virtual environment, the new description would have to be saved out and re-compiled before it could be tested.

CAEL-3D (Computer Animation Environment Language), is also a general language programming environment based on a large subset of Pascal.

Figure 4.9: CUBE-II program for converting fahrenheit to celsius

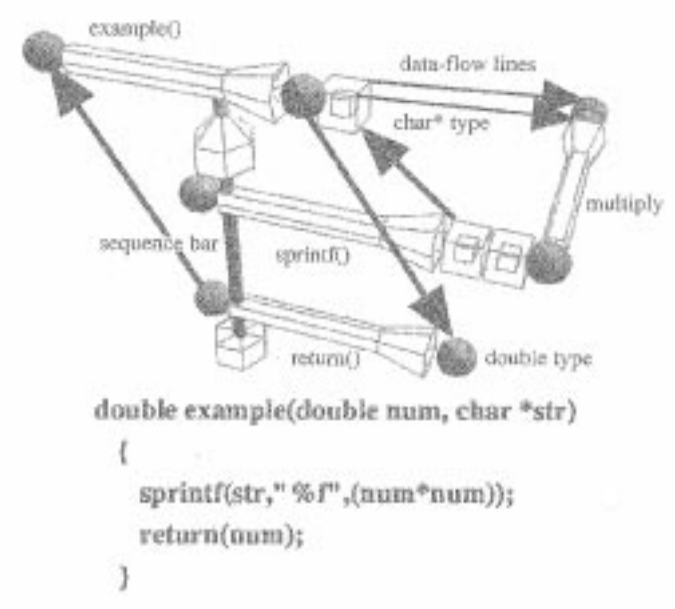


Figure 4.10: Lingua Graphica example

However the authors refine their application domain to that of animation construction since it exploits the 3D nature of the environment and enables a high degree of integration between the environment and application domain. Figure 4.11 shows the heading of a Fibonacci function.

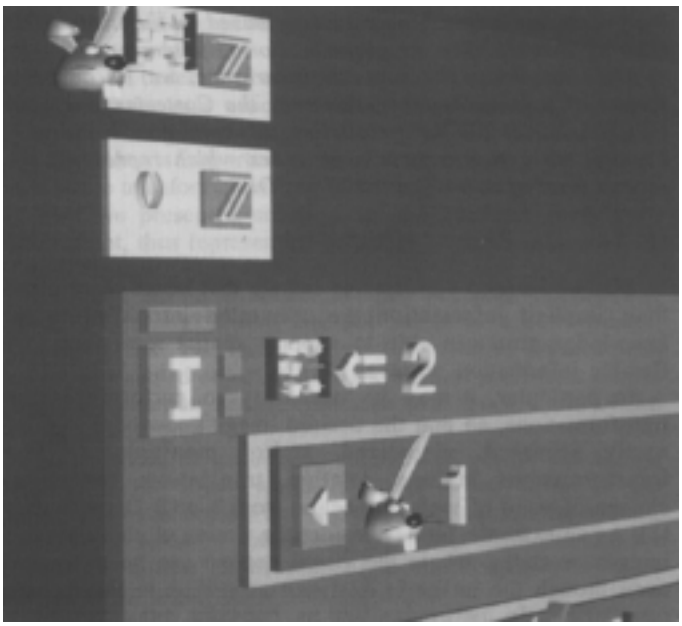


Figure 4.11: Program described in CAEL-3D

4.6 Immersive Description of Virtual Environments

Most current virtual environment systems provide a rudimentary form of customization of the virtual environment, in that the participant can pick up and move objects in the environment. This ability can be provided at little cost to the system designer since it relies on interactions that would normally be allowed. Specifying other parts of the scene description are harder since they involved data types that can't be mapped simply onto to object positions and interactions not usually provided in reality. Two examples are specifying the ambient colour and texture of an object which can be accomplished by the use of virtual tools within the environment, see [GB93] for examples.

Immersively constructing the geometry of the objects requires quite sophisticated virtual tools. A number of such systems exist. As examples; 3DM [BDHO92] provides a sphere creation function that allows simple objects to

be sculpted , Deering [Dee92] describes a virtual lathe demonstration application, and tools to create and manipulate surfaces have been constructed by Usoh and Slater [USV96]. The work described in Chapter 5 does not provide any object creation tools, though they would completment the system well, though as we shall see it provides a simple way to build the interfaces to such tools as would be needed.

To program behaviours and interactions of the virtual environment immersively would require a system that was the union of the two types of visual language from the previous section. It would be a visual language within a virtual environment to describe the virtual environment, that could interactively modify the appearance and behaviour of the virtual environment without the participant having to leave the virtual environment. dVISE from Division [Div94a] is an example of a system that goes some way towards such a system in that there is a menu driven interface within the virtual environment that allows editing of object properties and simple behaviours. Figure 4.12 shows an immersive menu with nine options. The centre object is the participants' hand. The menu items are (clockwise from top left), Lighting, Create Object, Modify Object Hierarchy, Modify Participant Characteristics, Save to File , Delete, Colour Editing, Spline Editing, and Stop Behaviour .

Figure 4.12: DVISE immersive menu

4.7 Conclusions

This chapter has reviewed the components that make up a virtual environment and reviewed work in visual programming that will be relevant for later discussion.

There are two techniques used in virtual environment construction: scene description and application coding. These two are far from disjoint, but geometry and object properties are most easily specified in a specially designed

language, whereas behaviour and simulation are usually coded in a programming language. The two can be linked together through an event mechanism that augments the scene description language with names of functions to call in response to events occuring.

Several paradigms of visual programming have also been reviewed with particular focus on data flow languages and systems that extend the metaphors of visual programming to three dimensions.

In order to describe immersive virtual environments a visual programming language should support the types of service provided by standard toolkits and again there was a separation between two possible levels of description: immersive scene description and immersive application coding of behaviours.

The next chapter will present a virtual environment within which it is possible to edit the behaviour and structure of the virtual environment objects whilst immersed inside the virtual environment. This is done through a 3D data flow description of object behaviour within the same environment as the objects it describes. The full range of services usually provided by a virtual environment toolkit are provided, and the results of editting the data flow can be seen without leaving to re-read or re-compile. In particular the virtual environment will allow interactions of the participant with the virtual environment to be specified allowing us to forfil needs for immersive interaction described in Chapter 3. Since the behaviour specification of objects and the objects themselves are presented in the same environment the environment acts as it's own visualization of program execution and program data which leads to tight debugging loop for new applications.

Chapter 5

Virtual Environment Dialogue Architecture

As described in Chapter 4, most virtual environment systems are statically constructed through configuration files and application code. This defines a fixed interaction metaphor and style for the duration of the experience.

There is no need for this interaction to be of a fixed form and this chapter describes a system that allows the participant to manipulate the dialogue structure that connects the input devices and tools that exist inside the virtual environment. In accordance with results from Chapters 2 and 3 this system does not impose constraints on the interaction metaphors allowed.

Essentially the virtual environment dialogue architecture (VEDA) system is a hybrid visual programming language and object hierarchy. The flow of data, from input devices through filters to object behaviours, has a visual representation that can be manipulated while inside the environment to have immediate effects on the environment. The elements of the data flow provide the functionality of the environment and are derived from consideration of the programming toolkits and environment description formats described in Chapter 4, as well as the tasks and modifications required by the work in Chapters 2 and 3.

Section 5.1 gives an overview of the abstract model used to describe an environment. The motivation for the design of VEDA is summarized in Section 5.2. The data flow model is described in Section 5.3 and the basic elements of the data flow are more fully explained in Section 5.4. The standard set of functions used in the data flow are described in 5.5 and Section 5.6 then gives examples of how these functions can be composed to produce higher level composites. A standard environment within which editing usually takes place is presented in Section 5.7 and finally Section 5.8 describes the implementation of VEDA.

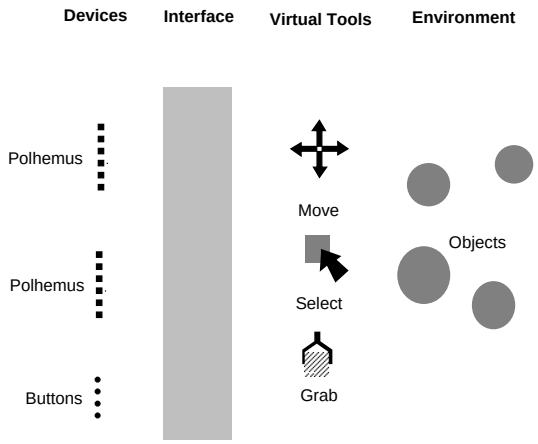


Figure 5.1: Components of the model

5.1 Overview of Abstract Model

By a dialogue architecture we mean the rules that determine the causal relationships consequent on gestures made by the participant in a virtual environment. The components of the dialogue architecture are shown in Figure 5.1. The first layer consists of a set of devices that return information about the position and state of the participant's body. Typical devices include Polhemus trackers that can return information about the position of a part of the participant's body and buttons that the participant can press. The virtual tool layer consists of a set of functions that affect the parameters of the environment objects or the environment as a whole. Virtual tool functions can, for example, manipulate object properties such as colour, position, velocity, sound and behaviour, and global environment properties such as gravity, scale and time. Each tool function requires a number of data elements as input in order to specify and perform it's task. For example a colour function would need to know an object to apply colour to and the colour to apply to it.

Mapping data returned by the devices onto the virtual tool function inputs is the task of the virtual environment interface. This contains sets of functions that process the device data streams or generate new streams that report events that occur within the virtual environment. Functions take input streams as do the virtual tools, but they generate one or more output streams as well. For example a filter that performs a logical and function would take two integer input streams, and generate a single output integer stream that was their logical and.

Within such an abstract structure the *fly in the direction of gaze* metaphor

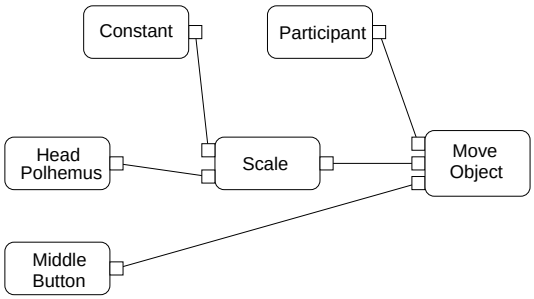


Figure 5.2: Definition of the fly in the direction of gaze metaphor

(§3.1.2) could be defined as in Figure 5.2. The components of this diagram are:

- The *move tool function* alters the position of an object in the environment. It takes three values as input: firstly a position value from which it extracts a direction vector to move along, secondly the identity of an object to move, and thirdly an enable node that activates the move function.
- The *head polhemus device function* returns the position of the participant's head in the virtual environment. This includes information about the direction the participant is looking.
- The *middle button device function* returns the state of one of the buttons on the 3D mouse that the participant is holding.
- The *constant device function* represents a virtual device, a slider for example, which generates floating point real values.
- The *scale filter function* scales a position value.
- The *participant object* is a component of the virtual environment that represents the participant themselves.

The lines represent the data stream connections that have been made. The data streams are strictly one-way and flow from left to right in the figure. The head polhemus device periodically generates the position of the head which is passed to the scale function. This scales the position by a constant value and passes this to the move function. The move function knows the identity of an object to move, but does not do so until the middle button function passes it a true value.

The VEDA system described in this chapter provides a representation of this dialogue architecture and allows the connections to be altered and new

functions and objects to be placed in the dialogue to re-configure the virtual environment.

5.2 Motivation

The abstract model is a data flow model and the obvious and most common representation is of a graph with nodes representing functions and arcs representing data stream connections.

The foremost reason for using an immersive representation of this dialogue was that editing could take place within the VE and not require the participant to repeatedly enter and leave the environment when making modifications. However our discussion of IVE systems in Chapter 3 highlights some features of an immersive 3D presentation that be of intrinsic value to the editing paradigm.

Firstly the immersive 3D workspace gives a larger working volume than a desktop one. Indeed since we will be editing the dialogue within the described VE the working volume is the size of the presented VE. We can also make use of any of the interaction techniques presented in Chapter 3 that support the sense of presence within the VE. In particular since many of the tasks involved in designing virtual environments are inherently three-dimensional, they may be best performed within a virtual environment. Defining full body gestures is a particular example of a task that would be difficult to perform outside an IVE, but even positioning and scaling of objects are two tasks that should benefit from naturalistic immersive performance.

Secondly the 3D arc and node display is a direct extension of typical 2D data flow representations which have been successful in many application areas [Hil92], and there is evidence to show that a 3D node and arc diagram may offer substantial benefits to comprehension of the dialog structure over the 2D diagram approach[WF94]. Within a 3D VE there are several new presentation capabilities available such as icons with 3D geometry, 3D animation and directional sound cues, that may help to overcome the naming problem that occurs when using many iconic desktop environments.

The choice of combining the VE with its own dialogue architecture gives several practical benefits to the editing process including:

- No loss of presence in the VE when editing.
- No loss of state in the VE. This avoids the problem that if the programmer exits and restarts the simulation, to test their modification may involve navigating and interacting with objects to regain the state they were in before.

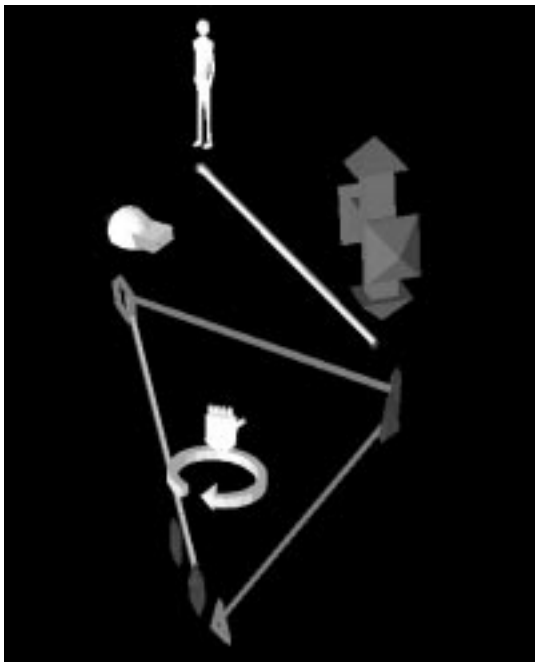


Figure 5.3: Immersive representation of the virtual treadmill metaphor recognizer

- No inconvenience of having to relate virtual objects to their database names.
- No loss of place in the VE. Re-entering the VE may mean that the participant has to re-orient themselves to the application environment. This complements the applications loss of state in that this is the participant's loss of state.

From a design point of view there might be another more subtle benefit to immersively editing behaviour in that since the participant has a sense of presence within the environment they are designing for, it may be the case that any techniques or design decisions that are effected would be more "appropriate". Essentially one would hope that since the participant is present within the VE anything they would conceive of designing or altering would be consistent with the VE's overall style.

An example of the immersive representation is shown in Figure 5.3. The components of this Figure are explained in Section 5.5.4, but it implements the Virtual Treadmill described in Section 3.7.

With such a visual representation we have created a programming language that has aspects of many of the types of visual languages described in

Chapter 4. The underlying architecture is based on a data flow paradigm and the immersive representation is a straightforward representation of this. It falls into the class of *visual programming languages* (§4.3.4), since the VEDA system is for actually programming, but since the environment within which the programming is performed is an elaboration of the actual environment it describes, it is also a *visual environment for the visualization of program execution* (§4.3.1). There are some elements of *visual coaching* (§4.3.2), since some aspects of the environment are demonstrated, rather than being abstractly defined. Gestures and animations are the main example of this.

5.3 Basic Data Flow Model

The basic building blocks of the behaviours are *function objects* that denote a function that processes streams of data. As mentioned before these fall into three categories depending on the combination of input and output streams that they process:

- **Devices** that return information about the participant, for example 3D positions of body parts and button states, and initiate streams of data. These data streams contain individual samples from the sensing devices, and are updated with new data at the sampling rate of the device.

- **Filters** that process data samples on one or more incoming streams and then generate new data that is propagated to the output streams.

- **Tools** that take input streams and act upon the objects of the VE. For example, one of the input streams to a colour function (§5.5.5) is a floating point value for the red component.

The function objects process streams of data, and each function object is associated with two sets of *data objects*, one of which maybe be empty. Data objects come in two flavours: *input* and *output*. Input data objects are holders for the data received from data streams and output data objects are the holders for outgoing data. A function object may have several input and output objects and when activated the function will use the current values in its input objects to generate output values that are then sent to the object objects [1]. These output objects are usually, but not necessarily, connected to

[1] In fact the input objects only represent the public interface to the object, in the implementation there is the provision for functions to have private state as well.

a number of input objects. These connections in fact form the *data streams* of the data flow model.

Data objects come in several basic types:

- integers
- floating point real numbers
- 3D transformations, 4 by 4 matrices of floating point real numbers
- object identity, and sets of object identities
- gesture patterns (§5.5.4)
- animation paths (§5.5.5)

There is no restriction on the number or type of data objects a function object could be designed, apart from considerations of the legibility and utility of the corresponding structure.

The association of function objects with data objects and their ordering is implied from the object hierarchy of the objects. Each function object has two *object sets* as sub-objects, each set containing zero or more data objects. The first set is the input objects, second the set is the output objects. See Figure 5.4.

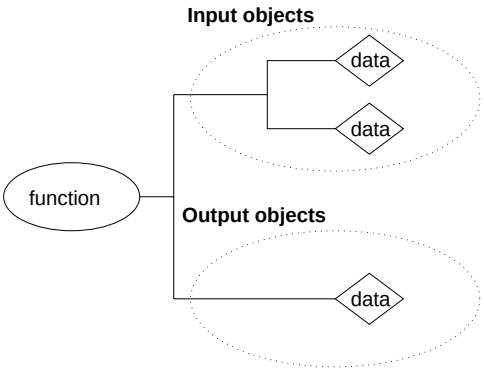


Figure 5.4: Function object with two input data objects and one output data object

The immersive representation of a *and* filter function which has such a hierarchy is shown in Figure 5.5. The *and* filter function is represented by a icon that depicts it's role. The data objects are represented by rings, with the number of sides depicting type and colour indicating their whether they

are input or output objects. There are no constraints on the juxtaposition of these objects in the environment, though usually they are clustered in two groups below the function they are associated with.

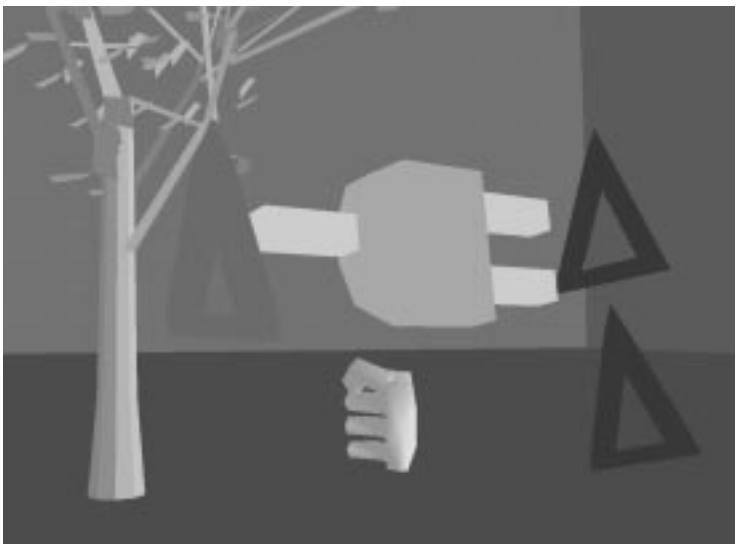


Figure 5.5: Immersive representation of an *and filter* function

Each output object maintains a list of destinations to which to send its values. Output data objects can send data to multiple destinations but input objects usually receive data from only one output object[2].

The flow of data through the connection network proceeds in three stages:

- Data is copied from output objects to input objects according to the output object's connection list.
- Once data is received at an input object, a decision is made, depending on the function of which it is a sub-object, of whether to call the function.
- If the function is called, it processes the input and puts the data into the output objects.

The flow of data is initiated by data propagating out from the device functions. The device functions simply place relevant external data onto their output objects.

[2]One case where multiple inputs are useful is when a functions can be fired asynchronously by two different events in the environment.

The firing decision depends on the semantics of the function that would be evaluated. For example, in the fly navigation metaphor, illustrated in Figure 5.2, only data being received at the enable input object (of integer type) causes the function to activate. This mechanism is used to improve performance and keep down the amount of data passing as examples in the following sections will demonstrate.

A major decision about the data flow model has been made in choosing this three stage data flow approach. This is that flow of data is *data-driven*, i.e. it is initiated by updates from external devices. An alternative architecture would be a *demand-driven* flow of data, where tool functions request data from their input objects in order to calculate their values. These requests propagate backwards, in the sense we have described, until all relevant data has been accumulated in order to calculate the function. In particular when the request reaches a device function the device is polled to find its current state. In this model, data flow is initiated by certain tool functions being activated periodically, usually on a per frame basis.

The advantages of this approach is that if the time to do the back propagation is known, values of model parameters can be obtained just in time for the display cycle of the environment and are thus as up to date as possible for rendering purposes.

The disadvantages are that polling the devices can introduce a source of lag, and that functions may be evaluated more than one per display cycle in order to obtain the most up to date value. Other complications for this method arise from the facts that the update rates for display may vary, as might the device data reporting rate[3].

The data-driven method was chosen primarily because the low level device polling is performed by a separate actor process which does not support direct polling from other actors (§5.8). However even if this restriction were removed, calculating the propagation time would be complicated by the fact the data flow structure can be changed interactively.

5.4 Environment Objects

The objects that appear in the environment come in three classes. The first two, data and function objects have been described and subsequent sections will outline in more detail the available functions and their representations. The third class of objects are *meta objects*. Meta objects serve to group together a set of objects, and to encapsulate them inside a higher level interface.

[3]Since the base operating system is a Unix, guaranteeing timing constraints is problematical.

A meta object has three sub-objects:

- **Components Objects**, a (possibly empty) set of function or meta objects. The components are elements of the environments behaviour and dynamics that are semantically related to the meta object's role within the environment. Thus an animated object might have as it's components the animation, filtering and triggering functions that generated that animation. This is not necessarily the case but provides a way of enforcing a structure and partitioning onto an otherwise complicated data flow diagram.

- **Interface Objects**, a (possibly empty) set of data objects that can serve as a higher level interface to the collective behaviour of the meta object's component objects. This mechanism allows us to create a meta object that appears to act like a function object, in that it has input and output objects, whereas it in fact serves as an encapsulation of several other functions. The interface data objects can be considered as the public connections for data streams. Data objects of the component objects whose data stream connections would be the same over all instances of the meta object are considered private and there is no need to place these in the interface.

- **Identity Object**, an output data object that references the object and serves to identify the object to any function that would operate upon it.

A typical structure is shown in Figure 5.6. The interface of a meta object is an arbitrary, possibly empty, subset of the input and output objects of any of its component objects. The main use of meta objects is to encapsulate several other objects and provide a higher level interface to their combined behaviour.

The possible nesting of meta objects within other meta objects lets us build an object hierarchy, and construct complex objects easily. Section 5.5.7 will describe tools for hiding and revealing part of the object hierarchy and Section 5.5.8 will describe the tools for manipulating and building the object hierarchies. Briefly, any part of an object hierarchy can be copied and deleted, and parts of the hierarchy can be hidden under certain conditions so that irrelevant detail does not clutter the environment.

In a typical application the environment will consist of a number of top level root meta objects. Some of these meta objects will be static objects with no behaviour and thus their component hierarchies will be empty, others will be active and have behaviours described by the objects in their component

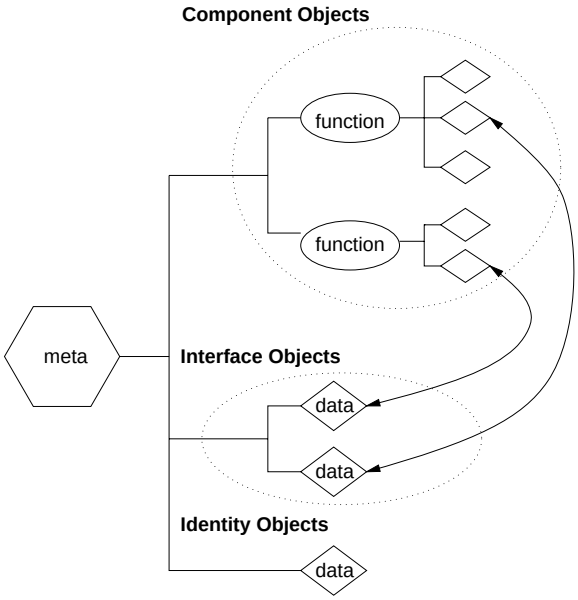


Figure 5.6: Meta object with two component objects, two interface objects and an identity object

hierarchies. In the default state where the application is being run and none of the interaction structure is shown, only the representations top level meta objects will be visible. For example in Chapter 6 we shall see a top level ball object that encapsulates the bouncing and colliding behaviours that apply to the ball. In the default state the ball just carries out its behaviour and none of the functions that create this behaviour are visible. When the ball is being edited these functions are visible within the environment and can be manipulated.

Meta objects can have an arbitrary subset of the component objects' input or output data objects as an interface. This interface is usually empty for top level objects, but otherwise meta objects are a useful mechanism to encapsulate a large amount of detail from the dialogue within a single component.

Two example immersive representations of meta objects are shown in Figures 5.7 and 5.8.

In both examples, the small sphere below the meta objects represent the identity object[4]. The object in Figure 5.7 is an environment object that does not have any components and would simply be part of the background of the environment. Figure 5.8 shows a virtual button object that has a collide filter

[4]These identity objects can also be hidden and aren't usually visible (§5.5.7).

Figure 5.7: Immersive representation of a meta object with an identity object, but no components or interface

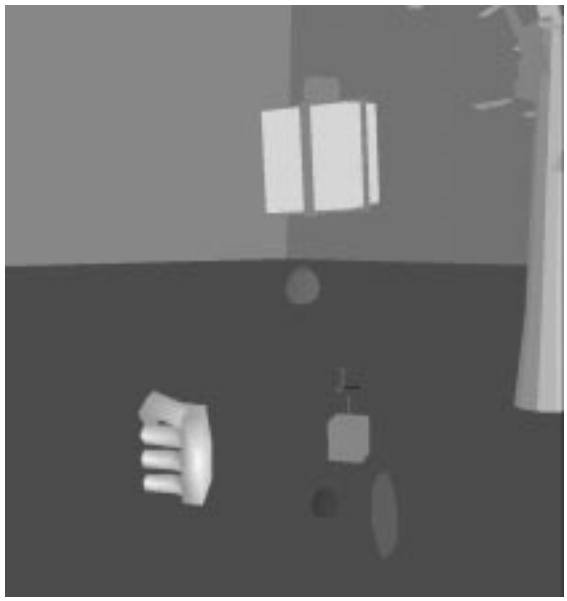


Figure 5.8: Immersive representation of a meta object with one component object and an identity object

function object as a component. The operation of the collide filter function object is described in Section 5.5.5, but the overall function of the button object would be to generate an integer stream that indicated when it was hitting another object.

5.5 Standard Components

The set of function objects and meta objects provided as a basis for constructing virtual environment has been designed at a certain level of abstraction from which it is simple to construct virtual environment applications. It is derived from considering the class of functions and properties provided by the standard programming toolkits and scene description languages described in Chapter 4.

The set of components is not exhaustive, for large applications it is inevitable that new functions have to be coded in order to create that environment. A complete list of the current functions supported by VEDA is given in Appendix A.

The motivating idea is that a large class of environments should be simple to build from the standard set of objects, and for more complex environments a prototype and framework can be built before application specific functions are coded.

The following sections describe the standard set of functions, meta objects hierarchies and design alternatives that were considered.

5.5.1 Participant Sensors

The first set of functions are the device functions that sense the participant's body posture. The VEDA system is currently built on a Provision100 VPX (§5.8), with two Polhemus trackers, one reporting hand position and one reporting head position. The hand tracker is embedded with a pistol grip that has five buttons see Figure 3.3.

A lot of detail is hidden within these functions. Determining the actual position of the head and hand is a complex procedure that relies on knowing several parameters, including: actual position of the tracker sources, the real position of the head and hand relative to the tracker sources and the virtual position of the tracker sources in the virtual environment. For example the trackers may be position 48 inches off the ground, so the reported position of the head should be the relative transform from the tracker receiver to its source transformed vertically 48 inches. Movement around the virtual environment for a limited distance is possible with just these transformations,

but for larger distances, a navigation metaphor must be used. The navigation metaphor effectively moves the virtual base position of the trackers in the environment. The reported head and hand positions are then the concatenation of these transformations and are reported through data objects of type 3D transformation. The decision to use complete transformations, rather than report three position and three rotation values separately, was made to reduce the number of objects in the data flow, the amount of data passing that would have to take place, and the amount of clutter that would result from all the parallel connections. Functions that take transformation streams as input often extract relevant information from the matrix, such as a point or X-axis vector, as a first step.

For convenience there is a third position device that reports the relative position of the hand from the head, in the head's local coordinate system. This will form the input for the gesture filter functions described in Section 5.5.4.

The state of each button is reported by separate device functions with single outputs, rather than a single device function with five outputs. This design was used because other platforms are likely to have different numbers of buttons on the controller. Indeed the original platform for VEDA, a Provision 200 system, had only 4 buttons 5.8.

A further device function is included amongst the participant sensor even though it does not represent any state of the participant. This is a time device that periodically updates a single integer stream with the current time. This is useful to drive certain filter and tool functions that rely on time varying values.

Figure 5.9 shows the representations of the device functions.

5.5.2 Standard Environment Functions

The basic environment can be edited using five tool functions, select, pick, copy, delete and connect.

The first four of these functions are standard functions that one would expect to find in any virtual environment editing system. The last, connect, is the means by which the data flow streams are connected within the environment. The immersive representations of the first four functions are shown in Figure 5.10.

Select The select tool function allows multiple objects to be selected so that further operations may be applied to them. This operates much like selection in 2D interfaces: when the selection button is pressed the object

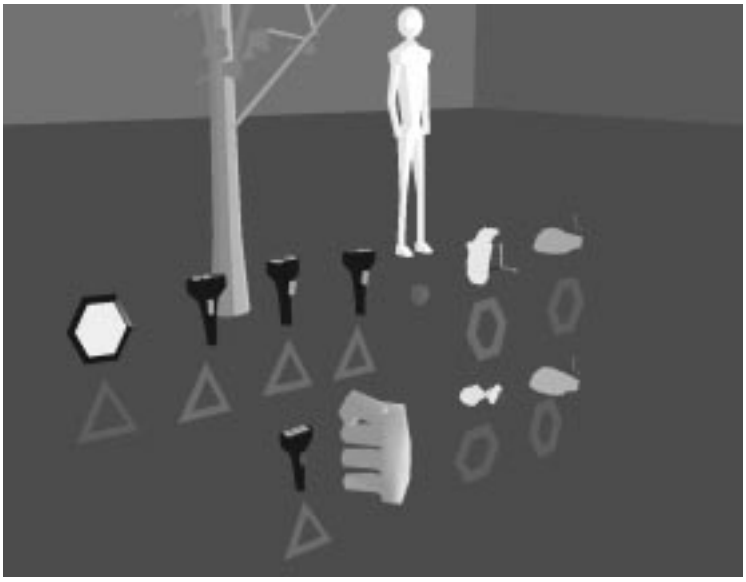


Figure 5.9: Immersive representations of the sensor devices

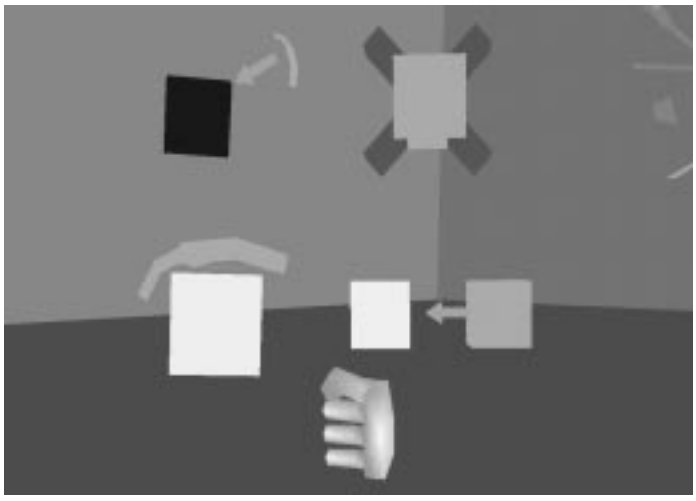


Figure 5.10: Immersive representations of the sensor devices

that is colliding with the selection device (usually the hand) is added to the selection set if not already in the set, or removed if it is already a member.

The select tool function thus takes 2 inputs: object to select, and integer stream that fires it. It has one output, the identity of an object set that holds the currently selected objects. Usually the object input stream would receive the object currently colliding with the hand, see Figure 5.11, though this is not necessarily the case. The selection set is indicated by each of it's objects being highlighted red until they are de-selected.

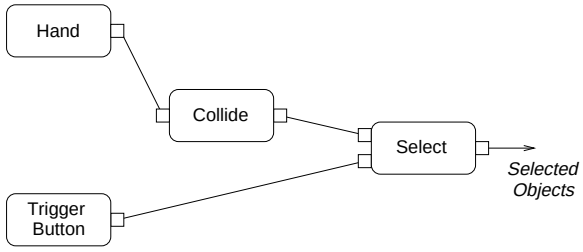


Figure 5.11: Abstract representation of the standard select tool function configuration

Many of the functions that act upon objects use the output object set of the select tool function in as a source of input to indicate the objects to operate upon.

Pick The pick tool function provides a way of moving a set of objects by enforcing a temporary constraint that the objects be attached to the hand. The pick tool function takes an object identity stream input and a integer input stream, and moves the objects in the set while the integer input stream is true.

Two obvious ways object input stream could be connected are:

- To the output of the selection set
- To the collision with the hand object

Figure 5.12 shows the two configurations by indicating with the dotted lines the two manners in which the object to pick can be connected. The first configuration, indicated by the lower dotted line, allows the whole set of selected objects to be picked simultaneously. The second, indicated by the upper doted line means that the hand must be colliding with the pick object.

As a metaphor for picking the first method has the advantage that multiple objects can be moved at once, but the disadvantage that these objects might not be local to the hand. The second method has the advantage that

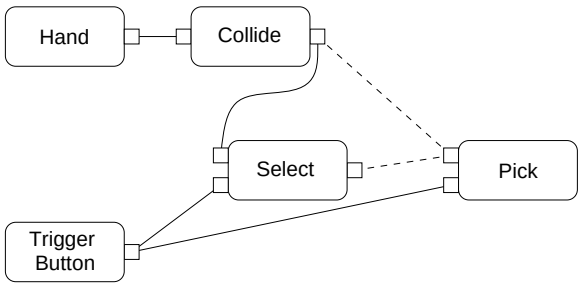


Figure 5.12: Abstract representation of the two options for pick tool function configuration

is it more intuitive in that the object being picked is touching the hand, but with some navigation metaphors where the participant must point with their hand to indicate direction, the object might obstruct the view.

A choice between these two configurations can be made and effected inside the environment.

Copy The copy tool function copies a whole object and it's sub-hierarchy. A choice of input configuration, similar to that for the pick tool function tool, has to be made, though it makes sense to choose a consistent metaphor. The copy tool function takes two inputs, object identity and integer and copies the object hierarchy, giving the new objects a positional offset, when the integer stream becomes true.

The copy tool function has to cope with object hierarchies and connections to and from the sub-objects. This is done in two stages, firstly the objects themselves are copied and secondly the connections are copied. When a connection is copied there are two cases to consider:

- Both end points are objects which have been copied. In this case the connection is made between the two new copies of the original end-points.

- Only one end point is within the set of copied objects. In this case the connection is made between the copied object, and the un-copied end point of the original connection.

This means that the copied objects mimic the original objects in their function, and relevant inputs will have to be reconnected. However the alternative, where no external connections are copied, could mean that a large number of connections have to be remade.

Delete The delete tool function deletes whole object hierarchies. In doing so it breaks any connections that originate at or are destined for any of the objects being deleted. It too takes two inputs, object identity and integer.

Connect and Disconnect In the current implementation these are not separate functions, but part of the pick implementation. Thus the connect and disconnect functions are not implemented individually within the dialogue architecture but their behaviour is hidden within and activated by using the pick tool function.

The pick tool function provides a number of *tubes* inside the virtual environment that can be used to connect data objects together. An unconnected tube can be picked at either end, and stretched with the other end remaining fixed. If the end of the tube comes within a snap distance of one or more data object one of three things will happen:

- If the other end of the tube is unconnected then the end being dragged snaps to the nearest data object.
- If the other end of the tube is connected, the tube snaps to the nearest data object of the same type. The tube will only snap between an input/output pairs of data objects.
- If neither of the above holds then the tube remains attached to the pick object.

When the tube lies between two data objects then a connection is made and the data stream initiated.

Disconnecting a tube from a data object involves either dragging it away from the data object it has just snapped to, or picking up the tube again. Picking up a tube along it's length brings the closest end to the pick object if the pick object's position is greater than the snap distance away from the data object, otherwise it remains attached.

This technique allows the data flow network to be constructed rapidly. The tubes provide a visualization of the data flow, and in combination with the level of detail tools to be described in Section 5.5.7 they provide a quick way of showing how an object is connected. Figure 5.13 shows the same *and* filter function as that of Figure 5.5, except that the input and output connections are shown. Both inputs have a single connection, and the output is connected to three separate objects.

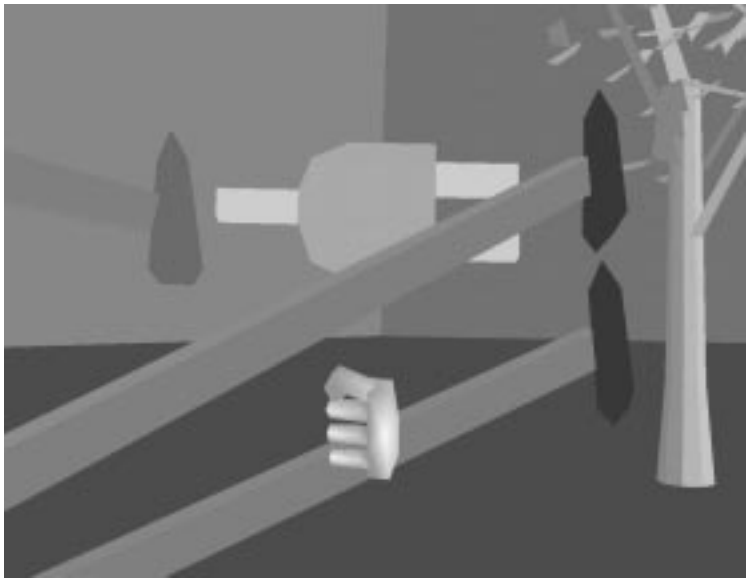


Figure 5.13: Immersive representation of the *and* filter function with connections illustrated by tube objects

5.5.3 Simple Filters

A set of simple functions provides integer stream filtering in order to construct more general rules about firing further events in the data flow. These are the *and*, *not*, *equal*, *then*, *double click* and *delay* filter functions.

The *and*, *not* and *equals* filters are simple functions that operate on integer streams and are analogous to logic gates. The *then* filter function is more complicated in that it takes two integer inputs and outputs success when the first goes to true and then the second after a set amount of time. This time is altered by a floating point real input. *Double click* is similar, but it takes only a single integer input stream, so it recognizes when a stream goes to true twice within the indicated period. The *delay* filter function simply delays an integer stream, by a time period set by a floating point real input value.

5.5.4 Gesture Recognition

As described in Section 3.2.3, many virtual environment systems provide gesture recognition, and this is provided within the dialogue architecture through types of node that represent train-able functions.

The nodes have to recognize two basic forms of gesture:

- *Static gestures* or *postures*. Here the node has to simply register when

a set of inputs stream take certain values.

- *Dynamic gestures* where the node has to recognize a pattern of data within a set of streams.

Of these two types of gesture, static ones are easy to report since they rely on a set of data elements falling within the ranges defined by a template. Such templates are easily demonstrated by making examples of the gesture.

Dynamic gesture are more complicated and Section 3.2.2 described methods of recognizing such gestures. Two methods of recognizing dynamic gestures are provided with VEDA: neural net based and feature based.

Neural Net Based Recognition The base neural nets implementation was simply incorporated from the work described in Section 3.7. A single neural net filter function accepts position values and returns a simple integer success value.

The neural nets can not be trained on-line so using one inside an environment requires a separate session where data is collected. The neural net filter function serves both the data collection and subsequent recognition services and the neural net is a data structure local to the function rather than a value that can be passed around the data flow network.

The neural net filter function object takes three inputs: a position stream from which the gesture is to be recognized, an integer stream that indicates when the gesture is being performed to provide training data to the neural net and an integer stream that stops the gathering of training data.

An initial true value on the first integer stream starts the recording of training data to an external record. Data recording is terminated by a true value on the second integer stream. Typically these might be temporarily connected to two buttons on the 3D mouse.

Once a neural net has been trained the function automatically starts recognizing the gesture from the values received on the position input stream and reporting on the output stream. The success and stop inputs are no longer required once recognition has begun and would be disconnected.

An abstract representation configuration of objects to recognize the virtual treadmill gesture is shown in Figure 5.14. The immersive representation of the gesture was shown in Figure 5.3.

Feature Based Recognition The approach to feature based recognition is similar to that proposed by Watson [Wat93a], in that we extract simple features from each individual stream and match them against the set of features that compose a gesture.

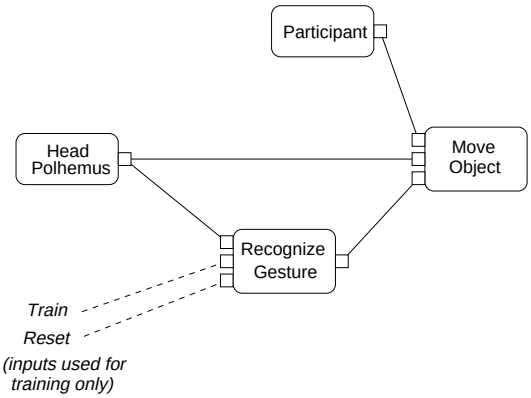


Figure 5.14: Abstract definition of the virtual treadmill metaphor recognizer

The features extracted are maxima, minima, start and end of stationary periods and start and end of gesture. A gesture is decomposed into feature sets for each of the of X,Y and Z axes of the position stream. Each feature set contains a list of features and the time ranges these should be recognized within, measured from the time of recognition of the start feature.

Recognition of the gesture is the successful sequential recognition of each of the features in each of the feature sets. Recognition for each of the axes progresses independently, though the start and endpoints should be reached at the same time and if any single axis recognizer fails to recognize it's next feature in the time limit required, all the recognizers restart. This avoids synchronization problems which might occur when an individual stream recognizer recognizes a repeated initial subsequence of the features.

A typical gesture sequence is given in Table 5.1. The gesture is that of making an alpha sign in front of the face.

Unlike neural net recognition, feature based recognition is provided by two separate functions. The first is a *gesture training filter function*, that has three input streams: a position stream from which the gesture is to be trained, an integer stream that indicates when the gesture starts and stops an integer stream to clear and initialize the gesture.

The first integer stream is required to become true when the gesture starts and remain true until it finishes. For each demonstration the set of features is extracted and merged into the current feature set using a modified longest common subsequence algorithm . More than one demonstration is required in order to set ranges on the times within which the features can be recognized.

The gesture training filter function outputs a description of the gesture which is passed to a *gesture recognition filter function*. This has two input streams: a position stream from which the gesture is to be recognized and a

X axis features					
	type	value (in)	range (in)	time range (s/20)	
first	start	2.3	0.8	0	0
second	end-flat	2.2	0.8	10	14
third	min	-9.0	1.5	26	32
fourth	end	3.7	0.3	24	27
Y axis features					
	type	value (in)	range (in)	time range (s/20)	
first	start	1.5	1.1	0	0
second	min	-10.0	1.3	29	35
third	max	-0.9	0.6	18	24
fourth	end	-5.9	0.8	11	16
Z axis features					
	type	value (in)	range (in)	time range (s/20)	
first	start	-19.5	0.9	0	0
second	end-flat	-18.3	1.0	47	54
third	end	-20.8	0.5	15	16

Table 5.1: Typical feature sequence of a gesture

gesture to recognize.

It performs continuous recognition and reports this in a single integer output stream. By using both nodes simultaneously it is possible to see whether or not enough examples have been given for recognition to be robust.

The difference in approach to the gesture training and recognition provided for neural net and feature based recognition is due to two factors: Firstly there is no need for neural nets to be passed around since this would only occur once (when the net was trained since there is no interactive training) and would be an expensive copying operation. Secondly, having the feature based gestures as a data type allows the possibility of using the gesture as an input type to a function that demonstrates the gesture so that someone can watch and learn how to perform the gesture. This isn't possible for neural nets since there is no way of decoding what the neural net recognizes.

Neural net and feature based recognition are suitable for different classes of gesture. Feature based is useful for gestures that have a path like description, such as iconic gestures, whereas neural nets are useful when the gesture is ill-defined and might vary from person to person. Neural nets can also provide continuous recognition in that they report success for the dura-

tion of the gesture, whereas feature based recognizers generate output when the gesture has been completed, and rely on their being an end-point to the gesture.

Static postures are easily supported by using a feature based gesture where the start and end points are the only two feature and they are coincident.

5.5.5 Object Properties

Various properties of the meta objects can be affected by functions within the environment. These property tool functions require the use of the identity objects as input to indicate the object to which they should apply and the are generally retrieved from the individual identity nodes of meta objects, or from filter functions that generate object identity outputs such as select or collide.

The property filter functions include *collisions*, *constraints*, *dynamics*, *animation*, *scale* and *colour*.

Collisions The first type of property is that of colliding with another object. There are two collide tool functions:

- **Specific Collide**. This takes two inputs, the identities of two meta objects, and generates a true value on its single output stream whenever the geometry of these two objects collide.

- **General Collide**. This takes a single identity input, and has two outputs, an integer value that it true when it is colliding and the identity of the object it is colliding with.

These two functions allow a lot of flexibility in defining reactions to collisions. The specific collide tool function could of course be implemented through the use of the general collide function and a node that test equality of the output object with the second object. However, the underlying collision detection algorithm (§5.8) is much more efficient if both objects are specified since it no longer has to test for possible collisions with every other meta object in the scene.

The virtual button of Figure 5.8 is implemented with a specific collide and it reacts only by being touched by the participant's hand.

Constraints Objects can be constrained together in three ways within VEDA:

- **To Constraint**. This constrains a slave object to follow a control object. When the control object is moved the slave is moved to maintain their relative position. However the slave's moving does not affect the control's position. The *to constraint* tool function takes two object identities as input: the first is the identity of the control object and the second is the identity of the slave object.

- **Along Constraint**. The *along constraint* constrains an object's position to lie along a 3D vector. The along constraint tool function takes two object identities as input: the first is that of an object from which the 3D vector's direction is taken, and the second is that of the object to constrain.

- **Between Constraint**. A *between constraint* constrains an object to lie between the two planes orthogonal and coincident with either ends of a vector. This is done by constraining the object's position so that it's projection on to the direction of vector lies between the endpoints of the vector. Again, the first input is the identity of an object from which the direction is taken, and the second is the identity of an object to constrain. This function also has a single floating point real output that gives a value between 0.0 and 1.0. This value corresponds to the relative position of the projection of the position of the constrained object between the ends of the vector.

Combinations of these three types of constraint allow quite a lot of flexibility in defining more general constraints. For example two objects can be linked together by using two to constraints. A constraint handler automatically resolves circularity in the graph of constraints. Section 5.6 gives examples of making a 1D slider and a 3D slider.

In the current implementation, no rotational constraints corresponding to along and between have been provided, though the required functions, *rotate around* and *angle between* would very similar to their positional counterparts.

Dynamics The object dynamics tool function controls the behaviour of an object that moves under gravity and responds to collisions. It requires five inputs: the identity of object to control, the identity of colliding objects, a floating point real value of gravity, an integer that acts as a reset to return object to it's original position and an integer animation time.

The function updates every time one of these values changes, and in particular at least as often as it receives an input from the second integer input. This is connected to a time device function that updates at a regular interval (§5.5.1). The identity of the object that the controlled object is

colliding with would usually come from a general collide tool function, so that the object could respond to collisions with any other object. The gravity value could be changed by connecting a virtual slider (§5.6), but once a suitable value was set this would probably be disconnected. First integer input returns the object to it's initial position when it becomes true.

Animation Animation of objects is provided by a pair of functions that operate in a complementary manner. The first is the *path create* tool, which creates a path from a sequence of input positions. The second is the *path follow* tool which produces position values which interpolate an input path at a chosen velocity.

The path create filter function takes 3 inputs: a position stream, a integer as a key point indicator and an integer as reset.

It has a single path output data object. It's behaviour is to add the input position to the output path whenever the first integer input is true. The second integer input simply removes the current path from the output stream.

It is complemented by the path follow tool function which takes 3 inputs: a path to interpolate, a floating point real that is speed of interpolation, an integer animation time and the identity of object to control.

The speed of interpolation is variable and allows control of how long it takes to complete the path. The animation time provides the necessary firing input to drive the animation at a constant rate.

Scale The basic pick tool function provides a way to position and arrange the objects inside the environment. To allow more general editing of the VE, a scale tool function was implemented that allows a meta object's geometry to be scaled independently along three axes. The inputs are three floating point real streams to use as scaling values, and the identity of an object to apply the scaling to. If the scaling is to be symmetrical about the three axes, they can all be connected to the same source. The provision of a higher level scale tool is presented in Section 5.6.

Colour The inclusion of a colour tool function completes the set of functions needed to customize the appearance environment objects. It's use is limited however by the fact that colour can only be applied to a geometry object as a whole, and not to individual geometrical elements. It takes three floating point real input stream that correspond to the red, green and blue components, and the identity of the object to apply the colour to. The provision of a higher level colour tool is presented in Section 6.3.2.

5.5.6 Position Filters

This collection of filters provide access to and manipulation of object positions. They might be considered object property functions, but they include functions to manipulate position data streams as well as set object positions. The filter functions are: *get position*, *set position*, *get relative position*, *set relative position*, *invert position* and *stretch position*.

The behaviour of each of these is quite straightforward. The get position takes an object identity as input, and generates an output stream with the object's position, whilst the set position takes two input streams, one the identity of a meta object and the second the position to move it to. Get relative position takes two position input streams, and generates the offset position from the first to the second. Set relative position, also takes two position input streams, and generates the concatenated position. Invert position simply takes an input stream and generates the inverse position on an output stream. Finally stretch position takes two object positions and generates a position which if applied to an object would stretch it between the two input positions. It effectively scales a one unit high object to the distance between the two objects, rotates it to lie in the direction of the vector from the first to the second position and then moves it to coincide with the first position.

5.5.7 Level of Detail

The amount of detail present in the environment description is quite large, so additional tool functions are used to hide parts of the object hierarchies that are currently unimportant. For example several levels of detail in the definition of a button are shown in Figure 5.15 and Figure 5.16. At the lowest level of detail the button is just an object within the environment. The next level up shows that it generates an output, and at the highest level we see that it's behaviour is composed of a single collide filter function and that it generates an output when an object collides with the button meta object. The dotted ellipse in Figure 5.15 indicates the detail that would normally be hidden.

All objects can be displayed at different levels of detail, with the constraint that function or meta objects that are not contained within a higher level meta object can not be completely hidden. This is so impossible for them to become irretrievable since invisible objects are uncollidable and thus unselectable. The levels of detail for each type of object are shown in Table 5.2.

Two tool function effect the changes between levels, *hide* and *reveal*. Their

Levels of Detail for Function Objects		
Lowest		.....Highest
Invisible and all sub-objects invisible	Visible and all sub-objects visible	Visible and all sub-objects visible at highest level

Levels of Detail for Data Objects		
Lowest		.....Highest
Invisible	Visible	Visible and all connections represented by tubes

Levels of Detail for Meta Objects		
Lowest		..... Medium
Invisible and all sub-objects invisible	Visible and all sub-objects invisible	Visible, interface objects visible and identity object visible
Medium		.....Highest
Visible, identity object visible and interface objects visible at highest level	Visible, interface objects invisible, component and identity object visible	Visible, interface objects invisible, component and identity object visible at highest level

Table 5.2: Levels of detail for each object type

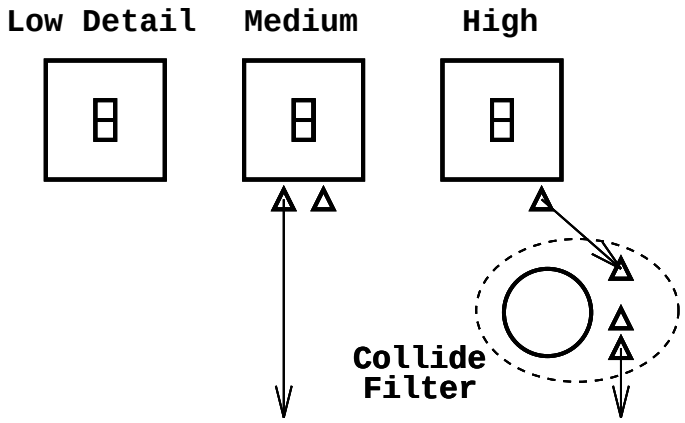


Figure 5.15: Abstract representation of detail of the button objects

representations are shown in Figure 5.17. Each tool function takes an object identity stream and integer stream as input. The reveal tool function increases the level of detail of the objects in the input stream by one, and hide decreases the level by one. Both functions are triggered when their integer streams become true.

5.5.8 Hierarchy Manipulation

Constructing the object hierarchy is a process of grouping together functions that apply to or are semantically linked to a meta object and forming them into a sub-hierarchy of that meta object. For example the participant meta object has all of the device functions as components.

Constructing this hierarchy is performed by two tool functions: *encapsulate* and *de-encapsulate*. Both tools can be used to manipulate both the components and interface objects of a meta object.

Adding component objects simply involves selecting a set of meta and function objects and adding into the component hierarchy of another meta object. Removing component objects is also a simple operation that removes an object from the component hierarchy it belongs to.

The addition and removal of interface objects is more complicated since the data object that becomes part of the interface can not actually exist at two places in the hierarchy. When a data object is added into the interface, a *ghost* data object of the same type is created and added to the meta object's interface object hierarchy. This ghost data object acts as a reference to the "real" data object, and connections to it are redirected to the data object. The ghost data object has a different position to it's real object, so that the interface objects can be grouped together with the meta object. To avoid

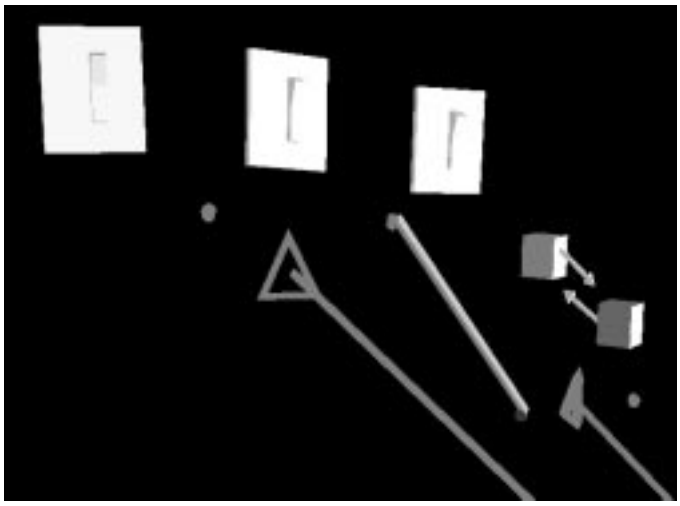


Figure 5.16: Immersive representation of levels of detail of the button objects

ambiguity a data object and it's ghost can not be visible at the same time, see the level of detail rules in Table 5.2.

Figure 5.18 shows the actual hierarchy of the button object described in the Section 5.4. The dotted curve in Figure 5.18 indicates the relationship between the button object's interface and the output data object of the collide filter function. The thick curve indicates an actual data flow connection. The connection indicates that the collide filter function responds to collision events of the button meta object.

Interfaces can be defined recursively, so the interface objects of a meta object can be included in the interface of a higher level meta object, and these ghost objects then form a chain and provide a mechanism by which the input or output of a function can be represented at several levels of an object hierarchy. Figure 5.19 gives an example of such a hierarchy. The dotted curves represent the relationships between the two interfaces. The data object can be accessed as an interface object at two different, depending on the level of detail of the various parts of the hierarchy.

Removing a data object from the interface involves deleting the relevant ghost object and reforming any ghost object interface chain it was a part of.

The encapsulate and de-encapsulate tools each fulfil two quite distinct roles, but because each of these functions are orthogonal in the effect they have on the object hierarchy, but are semantically quite closely related they are combined into one.

The visual representation of encapsulate and de-encapsulate are shown in Figure 5.20.

Figure 5.17: Immersive representation of the hide and reveal tool functions

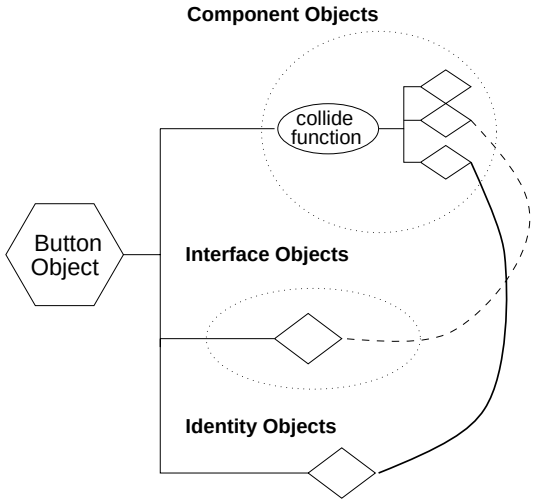


Figure 5.18: Object hierarchy of the button meta object

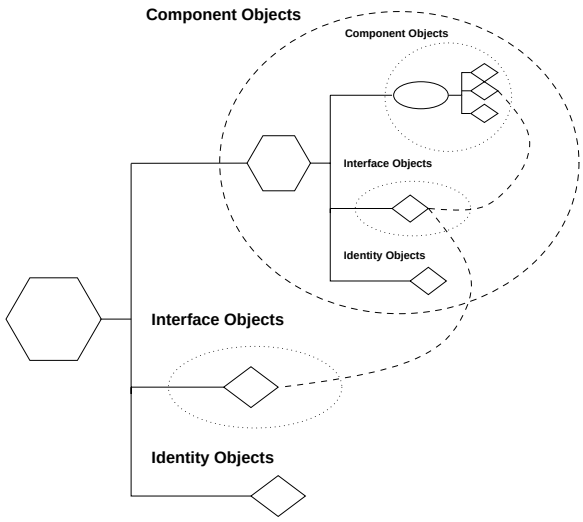


Figure 5.19: Object hierarchy of a meta object having nested interfaces

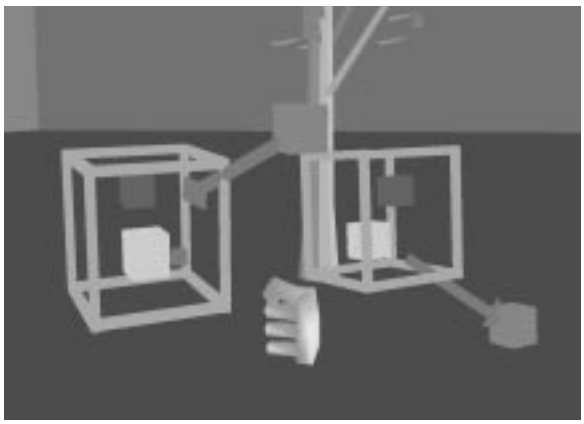


Figure 5.20: Immersive representation of the encapsulate and de-encapsulate tool functions

5.6 Example Composite Objects

Several meta object hierarchies have been constructed that can be used as basic components of new virtual environments. Virtual buttons have been described in Section 5.4, but others include: *virtual sliders*, a *multi meter* and a *scale tool*. A further example of a *colour tool* is given in Section 6.3.2 where its implementation was the goal of one of the evaluation exercises.

Each of these illustrates a different technique for generating meta objects with higher level behaviour, and they serve to show the flexibility inherent in the dialogue architecture.

Virtual Slider This meta object has a single output stream in its interface that generates floating point real values. Its overall behaviour is to act as the virtual equivalent of a real slider. It is a useful object since there are no real sliders attached to the hardware system, and several functions require floating point real values as input to control aspects of their behaviour. The virtual slider was constructed inside the virtual environment and is composed of the following objects: a meta object for the slider bar, a meta object for the slider button, a *constrain to* tool function that constrains the button to the bar, a *constrain along* tool function that constrains the button to lie along bar and a *constrain between* tool function that constrains the button to lie between the end points of the bar.

The abstract description of this is shown in Figure 5.21. The two objects are constrained together in three ways and the constrain between tool function gives a floating point real value between 0.0 and 1.0. This floating point real value is used as the output of the slider object and is placed in the interface of the slider bar meta object. The functions are hidden as components of the slider bar meta object.

The representation of the slider within the environment is shown in Figure 5.22. In this figure the interface is shown, but the components hidden.

Multi meter The multi meter is a debugging tool that has proved useful in the construction of application environments. It consists of two meta objects, a meter and a dial pointer, and it has several simple behaviours, depending on the inputs it receives. On receiving an positive integer input it flashes the colour of dial pointer. On receiving a floating point real input it moves the dial pointer relative to the meter, thus indicating the received value. On receiving an object input it flashes the colour of the object whose identity it receives.

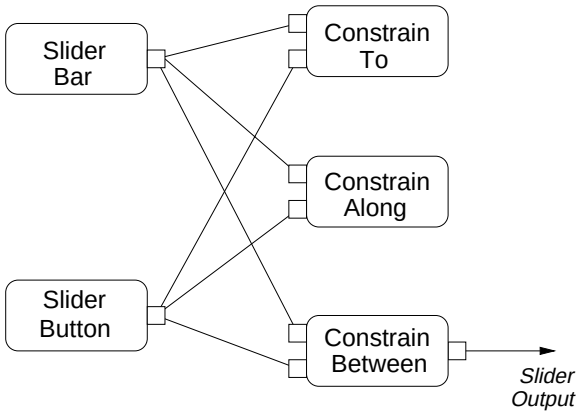


Figure 5.21: Abstract representation of the slider object

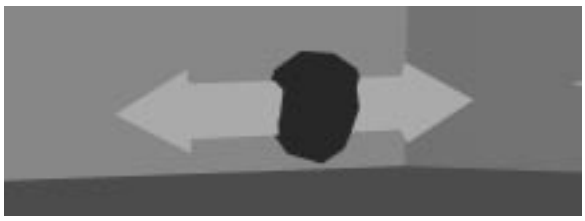


Figure 5.22: Immersive representation of the slider object

These behaviours are implemented as a single function[5]. The function takes five inputs, the identity of the meter meta object, identity of the dial pointer meta object and the three inputs given in the above list. The function is part of the component sub-object of the meter, with any combination of the input given in the above list as interface sub-objects.

Scale Tool Object The scale tool object provides a higher level interface to the scale tool function described in Section 5.5.5. It uses the interface hierarchy mechanism to hide most of the detail of the operation and so provides a useful tool for editing environments.

The representation is shown in Figure 5.23. This shows a meta object that represents the high level scale function. Attached to that is a slider object, and the only visible data object is a single object identity input.

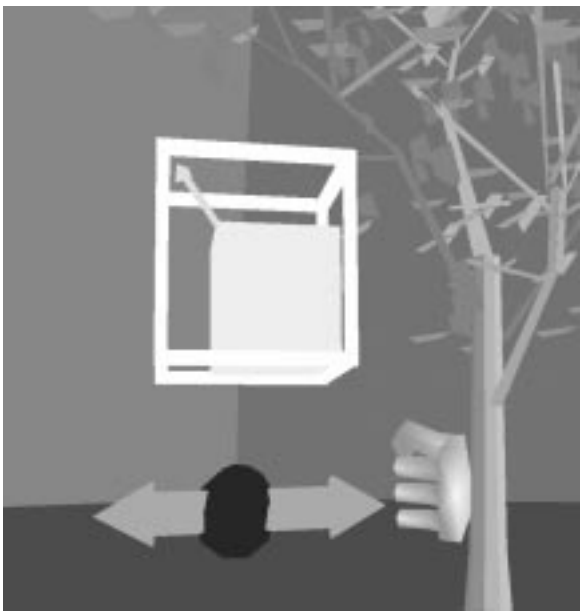


Figure 5.23: Immersive representation of the scale tool object

The abstract description of the dialogue occuring is shown in Figure 5.24. The slider object is constrained to the scale meta object. The output of the slider is connected to all three inputs of the scale tool function. The scale tool function is then made a component of the scale tool object, and the identity input of the scale tool function is placed in the interface of the scale tool

[5]This was because it was one of the first functions implemented since it was useful to test other functions, but now it remains for efficiency reasons.

object. The shaded elements of Figure 5.24 are those objects that appear at the usual presentation level of the scale tool object.

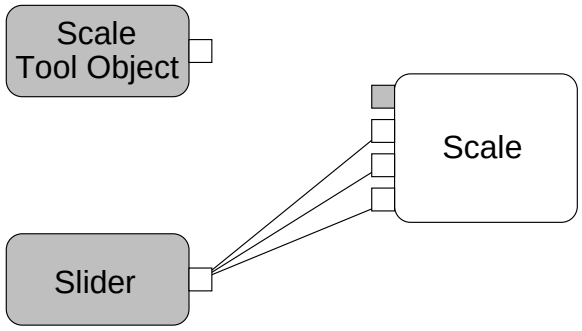


Figure 5.24: Abstract representation of the scale tool object

The scale tool object provides a symmetrical scale about all 3 axes of the object. It can be used by attaching the identity of the object to be scaled to the single object identity input in the interface of the scale tool object and manipulating the slider.

5.7 Standard Environment

The previous sections have described many of the functions and meta object provided with VEDA. All aspects of the dialogue are manipulable, but there is a standard environment and dialogue configuration within which development of applications and indeed the development environment itself takes place. This imposes a gross structure on the dialogue architecture, by providing a meta object hierarchy that hides most of the dialogue architecture.

The dialogue for an application can be separated into the *standard interaction techniques*, such as navigation, selection, picking and the dialogue editing functions and the *application interaction techniques* which are specific to the application.

The objects that compose the standard interaction techniques are arranged in two hierarchies, the *Participant* hierarchy and the *Tool box*, each of which is headed by a single meta object.

The participant hierarchy contains all the sensor device functions of the environment. The immersive representation of the participant hierarchy was given in Figure 5.9. The central human figure is the root meta object of this hierarchy.

The tool box contains all the tool functions and the dialogue that specifies their method of activation. The tool box and the collection of tools is shown

in Figure 5.25.

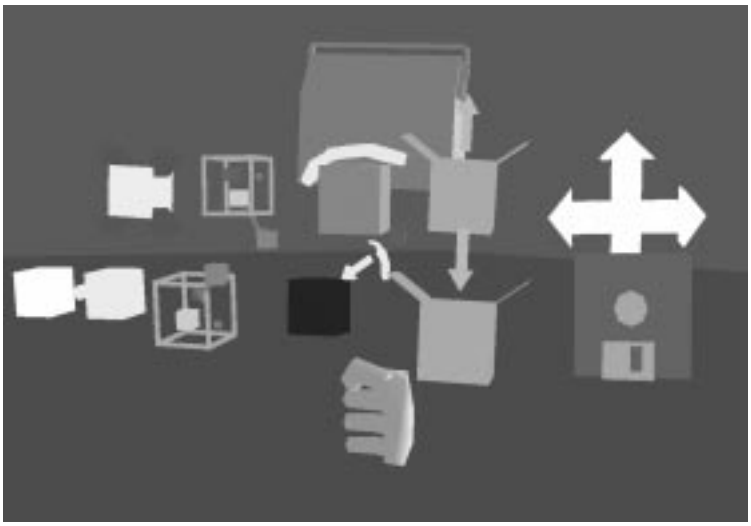


Figure 5.25: Immersive representation of the tool box and the contents

For application development the tool box contains a further object, the *object store*, which contains one of each of the filter and tool functions and various higher level meta object tools so that they can be copied into the environment and used as part of the dialogue. The object store is shown in Figure 5.26. Only part of the hierarchy is shown at a high level of detail.

Some of the higher level tools, such as the scale and colour tool objects, require the object to be present in the environment in order to use it. These tools need to be taken from the tool box in order to use them or the tool box needs to remain open. Any tool that has been taken from the tool box, or in fact any object, can be placed in the *tool belt*. This is not actually an object hierarchy, but a technique by which objects are constrained to the participant so that they follow the participant around. In the standard environment, the colour, multi meter and scale tool objects are in the tool belt, along with a tool that creates new tubes, the path training function and the gesture training function.

The objects that compose the application specific interactions are arranged in a set of hierarchies, the roots of which are the objects that will be presented within the environment. Thus each presented object will have associated with it parts of the dialogue that are semantically linked to that object.

Some overlap occurs between the objects in the standard and application interactions. A standard interaction technique, such as navigation, might be altered for individual applications. Also applications might involve the

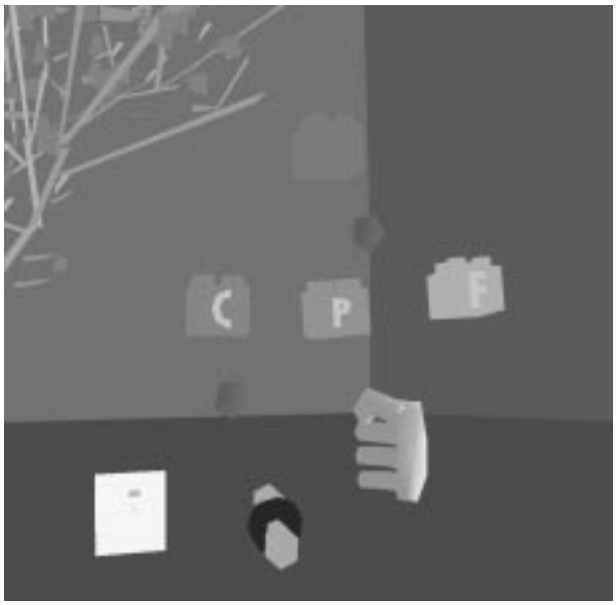


Figure 5.26: Immersive representation of part of the object store hierarchy

removal of some of the standard interactions. In particular once an application is finished a "runnable" version might be produced that has had all the unnecessary standard interactions removed, or at least disabled. Obvious candidates for removal or disablement are the level of detail tools and hierarchy manipulation tools. It is possible to set up a mechanism by which any disablement can be reversed by a knowledgeable participant through use of a special gesture or button.

5.8 Implementation

A first prototype version of the dialogue architecture was built for a Division Provision 200 running dVS version 0.2 [Div92]. This version of dVS differs in many ways from version 2.0 described in Section 4.2.1, though the basic actor model is similar [SS94]. The interaction devices used were a Virtual Research Flight Helmet and a 3D mouse with 4 buttons both tracked by Polhemus Isotrak devices.

A second prototype was built on a Division Provision100 VPX running dVS 2.0.4. This also had a Virtual Research Flight Helmet and a 3D mouse with 5 buttons, both tracked by Polhemus FasTrak devices. This machine provides much faster graphics since it uses a Pixel Planes 4 board for the rendering. The Provision100 is built around a 486 PC running Consensys

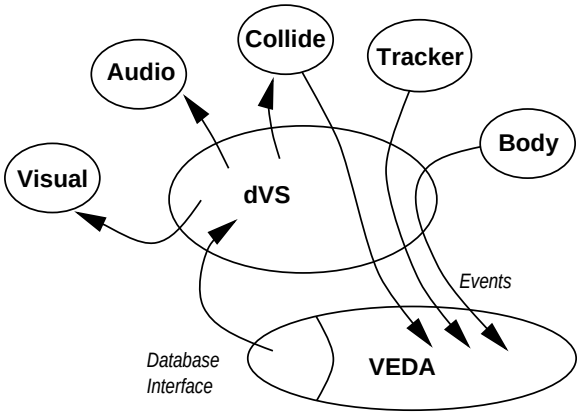


Figure 5.27: Relationship between VEDA and dVS

4.2 and the X11 version 5 windowing system.

5.8.1 dVS Services

The dialogue architecture was constructed by modifying dVS's application actor (§4.2.1). The process associated with this actor runs on the 80486 processor of the Provision and was written using the VC toolkit layer provided by dVS.

dVS provides many of the services required in the construction of VEDA. In particular it supports a VE object database which is shared amongst all of the actors. VEDA relies on the visual and audio actors for rendering of this database, the services of the collide actor for basic collision detection, and the tracker and body actors for device reporting. The relationship of dVS, VEDA and the various actors is shown in Figure 5.27.

The operation of the visual and audio actors is transparent, VEDA can only indirectly affect them by altering the scene database and re-specifying the viewpoint from which to render. All scene database operations pass through an interface which maps object property manipulations onto the requisite VC library calls.

The exception to this is the manipulation of the representation of the participant within the environment. This is provided by the body actor which directly accesses the tracker actor and provides a body description in the scene database. Navigation, which effectively moves the body representation through the scene, works by instructing the body actor to move the base coordinate system of the trackers, which has the effect of altering the reported positions of the head and the hand trackers. Since VEDA does provide navigation, it is necessary to turn off the body actor's own navigation

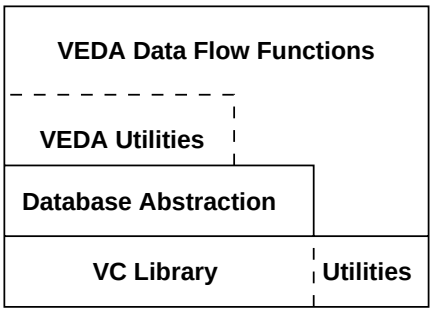


Figure 5.28: Library interfaces between VEDA and dVS

functions.

The initialization of VEDA thus involves loading a scene database, (§5.8.2), and registering certain event callbacks. The four callbacks are concerned with head movement, hand movements, button state changes and collision events. The arrival of these events triggers the flow of data through the dialogue architecture.

5.8.2 VEDA Database Layer

The standard component of the VEDA data base is the object, which encapsulates geometry, data and function. The database is written in an object-orientated manner, the three basic types of object, data, function and meta, can be considered subclasses of an abstract object class. Each contains a dVS visual object as its representation within the environment.

At a basic level VEDA provides database wrapper functions for dVS object property functions such as colour and scale setting, and utility functions to provide the extra functionality that VEDA needs such as logical hierarchy manipulation, level of detail, object creation and so on. Relevant information from other actors is passed using an event mechanism. This means that VEDA has little reliance on dVS being the underlying service, though some of the VC Library auxiliary functions (such as matrix manipulation and point to point transforms) are utilized out of convenience. The relationship of the libraries is shown in Figure 5.28.

The VEDA meta and function objects hierarchies are represented using trees of linked lists with, for example, a meta object having a subobject list containing 3 elements pointing to other subobject lists the first of which is the list of component, the second the interface objects and the third a unit list containing the identity data object. The data flow connections are represented in the data objects as a list of pointers to destination data

objects. Memory management for these structures is done locally with VEDA providing lists of unreferenced objects that can be re-used.

5.8.3 Data Flow Function Implementation

Each function object represents a C function which is triggered after data has been copied into one of its input subobjects. The function is called and passed the root of the data object hierarchy. Some functions are simply the data flow versions of certain VEDA utility functions such as copy and delete. The data object hierarchy will contain a single object, and this will contain the identities of the objects to copy. Others are the data flow equivalents of the database wrapper functions and the object hierarchy will be contain the identity of a VEDA object to operate on, and the corresponding property values to apply.

Other functions provide more complicated behaviour that is not provided or is limited in the underlying database. These include animation, dynamics, constraints and gesture recognition.

Animation is one aspect of the scene description language that dVS provides (§4.1). However it would interface badly with the VEDA system where paths can be interactively defined, since a MAZ spline path description is statically defined in a external file. Within VEDA it is possible to drive the animation based on arbitrary conditions since each update along the path is governed by a speed and is triggered by a data flow input. At the moment animation is linear between control points of the path.

Dynamics are also possible with dVS, but do not provide for flexible bouncing behaviours. To calculate dynamics requires precise knowledge of the position, velocity and colliding properties of the objects in question. However it is not possible to directly extract this information from the dVS database since updates are made asynchronously to the database and there is some latency involved. Thus each VEDA database object must maintain a current position and velocity, and since collision information is conveyed by events only, VEDA maintains its own collision database that can be queried by any VEDA function.

Constraints are also provided by dVS in a limited way in that an object can be constrained not to move or rotate about one or more of the local coordinate axes. The more general constraints that VEDA provides are implemented at the database layer. Whenever a filter attempts to set the position of a VEDA object a check is made against a set of currently active constraints, and any corresponding constraints are applied. The constraint functions themselves simply add and remove the various types of constraint from this set, and their application is automatic.

The gesture recognition functions are simple interfaces to external libraries. There are however hooks in the VEDA database layer to create, destroy and copy the gesture data types, since their size isn't static.

Chapter 6

Evaluation

The dialogue architecture described in the previous chapter provides an immersive environment within which it is possible to manipulate and create interaction techniques and behaviours of objects. Several examples of the construction of object hierarchies and data flow structures were given to illustrate the range of possibilities that are possible with the components supplied.

This chapter evaluates the VEDA system for the implementation and prototyping of environments. Firstly, in Section 6.1, it considers the requisites and desires for the manipulation of interactions that were the product of Chapters 2 and 3 and shows how this system provides a simple way to make the proscribed changes from within the system.

Secondly, in Section 6.2 we consider a moderately sized application that was constructed using the tools described in the previous chapter in order to show how the dialogue architecture gives a large degree of flexibility in the design of the virtual environment and how an environment affords different manipulation capabilities to participants with differing experience of the toolkit.

Thirdly, in Section 6.3, we consider applications built by novice virtual environment designers that illustrate the breadth of the toolkit, the ease with which interfaces can be prototyped and the advantages that such an approach gives, even if the desired behaviour falls outside the scope of the filters provided and requires new filters to be programmed.

Finally in Section 6.4 we place VEDA in a taxonomy of visual programming languages to illustrate the design decisions that were made.

6.1 Manipulating Interactions

Chapters 2 and 3 described work to design appropriate and intuitive metaphors for interaction with virtual environments. The conclusions of these chapters pointed out some shortcomings of current VE systems and voiced a desire for certain major and minor aspects of the interaction to be made modifiable for particular tasks, environments and users.

Some of the main technical points were as follows:

1. Changing the sense in which a toggle or switch works.
2. Redefining effects of button combinations or providing new combinations.
3. Defining gestures within the environment.
4. Reconfiguring or enabling gestures for interaction.

The first two items result from the work on the Desktop Bat, see Chapter 2, and the second two from the Virtual Treadmill metaphor, see Chapter 3.

Providing these types of modifications is simple to perform within the environment given the filter and tools provided in the VEDA system. The sense in which a button works can be inverted by using a *not* filter, and button combinations can be re-arranged using *and* filters. In fact the simple filters described in Section 5.5.3 allow a very broad range of dependencies on boolean events to be defined.

Gestures can be demonstrated and recognised within the environment using the two types of gesture filter given in Section 5.5.4. Again using the tools in Section 5.5.3 and the general data flow editing tools, many gesture dependencies can be defined and modified within the environment.

Reconfiguring interaction techniques is also simple with VEDA. Chapter 3 referred to the following three choices for the navigation technique in a virtual environment:

1. Fly in the direction the hand is pointing by pressing a 3D mouse button.
2. Fly along line of sight by pressing a 3D mouse button.
3. Fly along line of sight by making the gesture of "walking on the spot" (Virtual Treadmill metaphor.)

The data flow configuration for the second metaphor was given in Figure 5.2 and the configuration for the third in Figure 5.14. The difference between these two was the use of a gesture to trigger the *move* tool which is

an easy modification to make. The difference between the first and second metaphors is simply the direction in which the navigation occurs, and this can be changed within the environment by detaching the data stream from the head polhemus and attaching it to the hand polhemus[1].

6.1.1 Discussion

There is a large range of further navigation and interaction techniques that can be exploited. Some of these were discussed in Chapter 5. Some examples are:

- Cross-hair flying where the participants flies in the direction indicated by the vector between their hand and their eye [Min95].
- Selection at a distance [Min95]. Apart from reconfiguring the method of picking (§5.5.2), it is possible to make the pick object larger and use it to select objects far away.
- Movement constrained to a ground plane. Using the constraint tools described in Section 5.5.5 the participant can be constrained to a plane or a line.
- Virtual vehicle movement. It is possible to provide simple virtual vehicles in the environment to which the participant can attach and passively follow around, or more sophisticated vehicles which have their own virtual controls which the participant can operate.

The choice between these metaphors should be determined by the participant and the tasks which they are expected to perform. It is important to note though that although there is a tendency for the interactions described in this work to be based upon the body-centred interaction paradigm (§3.10), this is an application level design decision and many other styles of interface, such as menu based, are possible.

The approach to interaction definition within VEDA is highly modular since once a set of gesture functions has been defined or configured they can be replicated and combined, using the simple filters of Section 5.5.3 to provide more complex gestures. This can include posture sequences such as those described in Section 3.2, or gestures involving several body parts.

[1]This illustrates one problem in that during the reconnection the participant is unable to move unless a second navigation metaphor is enabled. For this reason the head and hand polhemus are close together in the standard environment.

6.2 Example Application

This section illustrates the components and possibilities for modification of an example table tennis application. The application allows play against a computer opponent or with, some modifications, against another participant in a collaborative set-up [SS96]. A view of the application is shown in Figure 6.1. In this figure, all of the data flow components are hidden within top-level meta objects in the manner described in Section 5.4 and the application is shown in "run" mode.

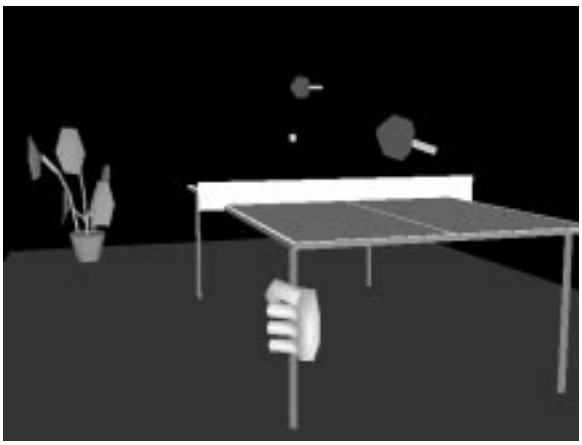


Figure 6.1: The table tennis application

6.2.1 Components

The major components of the table tennis application, the top-level representations of which are all visible in Figure 6.1, are:

- The participant - encompassing all the standard interaction techniques.
- The ball which has a dynamics behaviour described in full below.
- The bat and table on which the ball will bounce.
- The net and floor, collision with which will reset the ball to a starting place.
- The opponent, either an automaton or a second player equipped with a second bat.

Each of these is a meta-object inside the environment that contains several data flow objects. The data flow definition of the ball meta-object is illustrated in Figure 6.2. Two major components are shown, a dynamics behaviour receptor and a collide filter. The basic dynamic behaviour receptor takes five inputs: identity of an object to control, notification of the identities of colliding objects, a reset flag, time values and a gravity strength value. The reset input initialises the dynamics behaviour which in this application corresponds to serving the ball. The ball meta-object is connected to the identity input of the dynamic behaviour as would be expected, and similarly the dynamic behaviour is concerned with collision events with the same object.

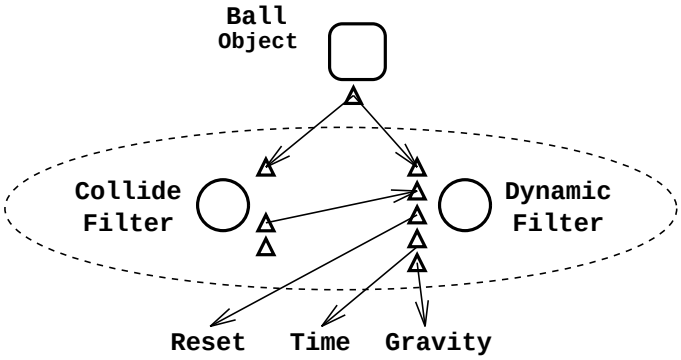


Figure 6.2: Definition of the ball object

Everything within the dotted ellipse of Figure 6.2 is a component of the ball object and all objects within and arrows leading to that ellipse are detail that would normally be hidden during play.

Figure 6.3 shows the immersive representation of the ball object and components. The layout is purposefully similar to that shown abstractly in Figure 6.2 but in this case external connections are not shown since they are connected to filters or other meta-objects that are currently not shown at a high level of detail.

The other major components of the environment, the net and floor are in effect buttons that are connected to the ball's dynamic behaviour reset input in order to place the ball back into the serve position.

6.2.2 Customization

Possibilities for customization are many and various. Any of the objects can be changed and copied so there could be two balls shaped of any shape. Thus table tennis is a very specific description of the application since the com-

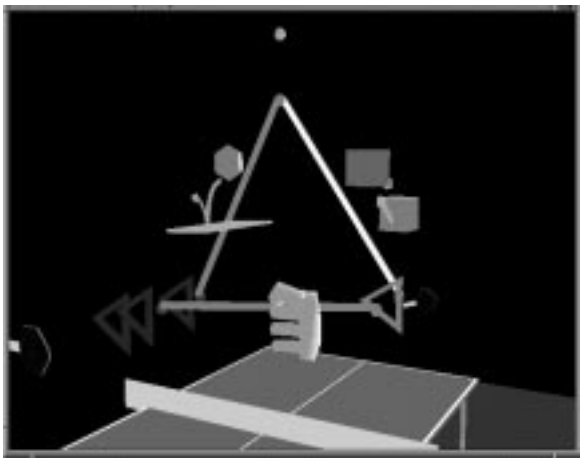


Figure 6.3: Immersive definition of the ball object

ponents could be re-arranged and modified to play volley-ball for example, though editing of the geometry of the objects themselves is not yet possible within VEDA.

More interesting customizations arise when considering how the ball is served, that is, determining what is connected to the reset input of the dynamic behaviour filter. Three possibilities are: connection to one of the 3D mouse buttons, connection to a virtual button and connection to a gesture recogniser. The second and third allow a lot of flexibility, for example the button could be positioned near to the table or could be the table itself so tapping the table would serve the ball. Alternatively, since gestures can be defined inside the environment, serving could be enabled by waving the bat under the ball or standing in the serving position.

Another customizable component of the application is the behaviour of the opponent. The core of the opponent is an application specific filter that knows about the ball and table objects and moves a second bat in order to return the ball. In a collaborative setup customization could involve simply removing this behaviour and having a second participant pick up another object and use it as a bat. The second participant could always cheat by not removing the computer player and play two against one.

One part of the application that is designed to be customizable is gravity. This is usually connected to a virtual slider since no real dials or sliders are connected to the current system. This slider might be part of the default application or might be hidden inside the component hierarchy of a meta object, in which case it would only be accessible to experienced VEDA users.

6.3 A User Study

A user study was conducted to evaluate whether participants could learn VEDA's model and use it to perform useful modification and creation tasks. This was an in-depth study and the three people who participated all accomplished tasks of significant complexity.

The format of each evaluation was three 2-3 hours sessions using VEDA, though one participant spent a considerably longer time using VEDA and developed an experiment for use in an MSc project.

Each of the participants had some experience of VE technology before and all had degrees in computer science. Participant A, a recently finished undergraduate had some experience writing scripts for the POVRAY ray tracer to describe and animate objects. Participant A had three years experience programming in C, and had completed a course in object-orientated programming in C++. Participant B was a second year PhD student studying artificial intelligence who had four years programming experience. He was interested in using AI techniques to describe VEs but had little experience with description languages apart from some experimentation with VRML1.0 and knowledge of the DXF format. Participant C was undertaking a MSc course in VEs, with a scene description element. He was undertaking a project using VEDA and had two years of experience with programming, though none using an object-orientated language. None of the participants had been exposed to a data flow language.

Sessions roughly broke down into one practise session to gain familiarity with VEDA, one experimental session where they would use the techniques they had learnt to do a few simple tasks and one exploratory session where they tried to accomplish a task that they had suggested.

Three rough classes of modifications were investigated: modification of an existing application, exploration and creation of new tools, and extension of VEDA with a new filter. These cover many of the same tasks that were performed in Chapter 5, and again they demonstrate the flexibility of VEDA for describing environments.

It was expected that each participant would be able to make simple modifications to the applications within VEDA. Given the short amount of time that they would have to become familiar with the set of filters available they were not expected to build radically new behaviours without some consultation. The criteria for success were that the participants would be able to access and understand existing data flow structures, manipulate the data flow using the tools discussed in Chapter 5, and be able to design and build simple new dialogues involving roughly six filters.

Overall once the participants became familiar with the environment and

reached a certain level of expertise with the tools and metaphors for interaction they found it simple to modify the dialog. Their accomplishments exceeded expectations as they found the data flow metaphor easy to understand and their suggestions showed quite a deep understanding of what could be accomplished with the tools and filters they had utilised.

6.3.1 Application Modification

Participant A did not have any specific goals when starting the sessions. The scope of the familiarization session was thus fairly broad and covered all of the major interaction techniques. This began with a familiarization with a table tennis environment without any of the modification tools embedded within it. This introduced the default navigation, selection and picking metaphors (§5.5.2).

Once comfortable with these tasks, the VEDA specific techniques were introduced. In rough order of introduction, these were copying and deleting objects, revealing and hiding levels of detail, connecting and disconnecting streams of data, and encapsulating and de-encapsulating objects.

Experimentation with these techniques involved the participant's accessing, decoding and modifying the behaviour of the ball. He gained access to the gravity slider, by revealing the ball's behaviour and decoding the data flow hierarchy. He removed the slider from the ball hierarchy using *deecapsulate* and could then modify gravity whilst the ball was bouncing. Once a gravity value was set the slider was replaced in the ball hierarchy using encapsulate and hidden away. Subsequently the participant revealed the slider and deleted it from the environment.

Initially the participant found the amount of detail in the ball description daunting, but given the 3D layout he was able to discriminate and trace individual connections. This was a common hurdle with all the trials, but more experience and familiarity with likely layouts overcame this. This particular participant did suggest making a new meta object which would have the ball's dynamics and collide behaviours as components, and an interface composed of the reset and gravity inputs of the dynamics. Figure 6.4 sketches the hierarchy that was proposed . This hierarchy would have the benefit that detail that does not change very often, such as the dependency on an animation time input and the collision detection, wouldn't be seen when more common modifications were made.

There were three main modifications made to the table tennis application: direction of navigation, the addition of a backwards navigation, and revision of the object picking metaphor for a specific object.

The process of making the first modification was described in Section

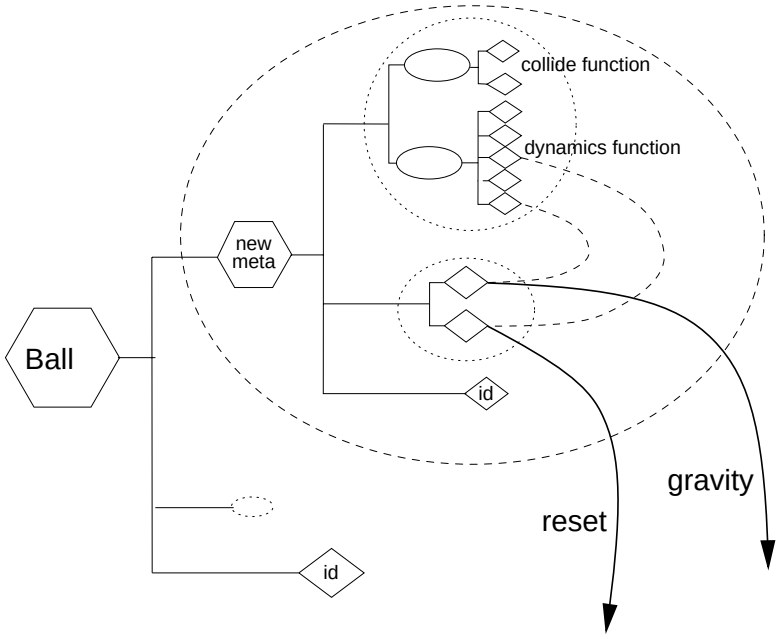


Figure 6.4: Proposed revision of the ball's objects hierarchy

6.1. The participant knew of this technique from the introductory literature and wanted to try it since he was uncomfortable pointing in the direction of movement. The initial attempt to disconnect the direction input from the hand sensor function and connect it to the head sensor function resulted in his being too far away from the head function's output to make the connection without putting his hand behind the tracking device[2], but he realised he could reconnect it to the hand and then move to a better location from where it would be possible.

However once this modification was made, a second was required since it was impossible to move backwards easily. Backwards motion was effectively a second navigation metaphor very similar to the first, except that it was attached to a different button, and the direction of motion was extracted from the inverse of the reported position of the head using an invert position filter (§5.5.6). The complete data flow for this navigation metaphor is given in Figure 6.5.

The third modification involved replacing the picking metaphor to make it easier to handle the bat when playing the game. The participant thought that having to hold the button in when holding the bat was inconvenient as

[2]A Fastrak only reports position in a half-space relative to the tracker source, the effect of placing the tracker receiver behind this half-space is to invert the reported coordinates and thus the hand "disappears" from view.

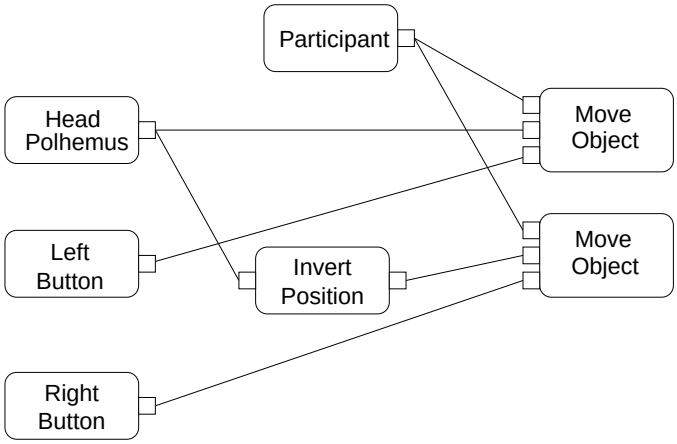


Figure 6.5: The complete revised navigation metaphor

he forgot occasionally and dropped the bat whilst making a movement. The first attempt at implementing this is shown in Figure 6.6a. This differs from the standard metaphor shown in Figure 5.12, in that the bat does not have to be selected and is automatically picked when the trigger button is pressed, and that there is a toggle filter between the trigger and the pick function. This means that participant simply has to press the trigger once to pick the bat up and a second time to drop it.

This metaphor worked very well, but given the behaviour of the pick function, (§5.5.2), a result was that when an object was picked the relative transform to the hand was maintained, and the bat did not always appear to be attached to the hand. This was remedied by explicitly setting the position of the bat object to the position of the hand, see Figure 6.6b.

Lack of time prevented any further modifications, but the participant was satisfied that the revised table tennis application was easier to use than the original one. Further suggestions made by the participant in the final session were to revise the navigation metaphor again to be cross-hair flying (§6.1.1), and to continue the modification of the picking metaphor. His suggestion was that rather than have picking enabled by pressing a button, the bat would picked up by touching the hand to it and put down by touching the bat to a new table object that would stand adjacent to the main table tennis table. The participant was confident that he could make these modifications given more time.

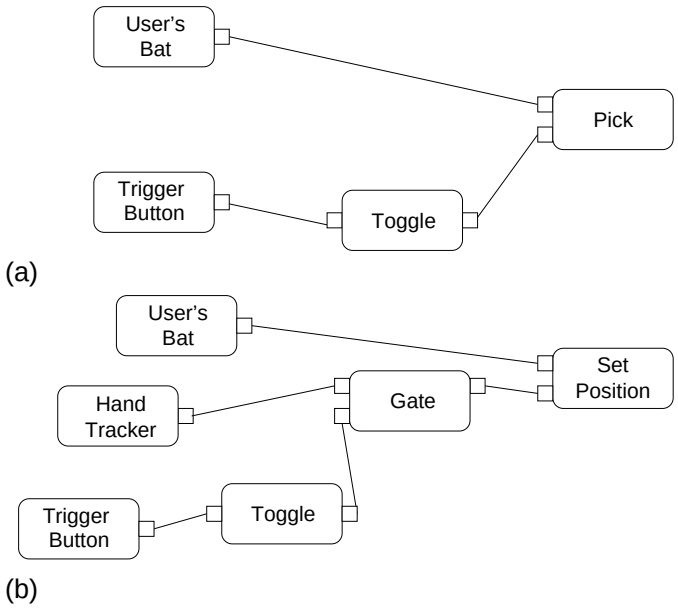


Figure 6.6: Two iterations of the bat picking metaphor

6.3.2 Tools Exploration

The familiarisation sessions for participant B followed a similar format to those of the first since he also did not have a specific task in mind. The experimentation phase also started with investigation and decoding of the behaviour of the behaviour of the ball. However this participant was side-tracked by the desire to change the scale of an object into investigating the behaviour of the scale tool (§5.6).

The participant found the complete data flow description of the scale tool initially confusing though once he had discovered what parts of the hierarchy did, he learnt to hide them away.

For the exploration phase it was suggested that he try and build a similar tool for colouring objects based around a colour function (§5.5.5). This takes as input the identity of an object to colour and three floating point values that represent the red, green and blue intensities to apply to it.

To explore how this function operated, the inputs were connected up directly, with one slider being attached to the red input, and the id object of a meta object directly connected to the identity input. Then the slider was copied twice more so that all three colour components could be specified independently. The next stage was to provide a simpler way to identify the object to colour. This was done in an identical manner to the scale tool, the object to colour is that which is colliding with a top-level meta object that

represents the new colour tool. An outline of the components of the resulting tool is shown in Figure 6.7a. The shaded elements of Figure 6.7a are those objects that appear at the usual presentation level of the colour tool object.

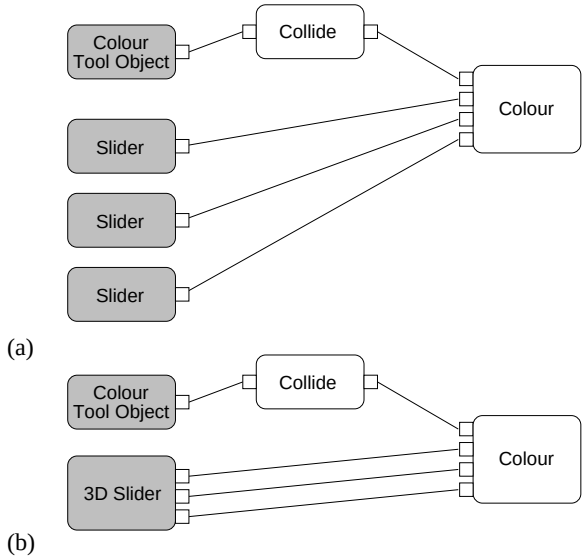


Figure 6.7: Abstract definition of two versions of the colour tool object

The three sliders would usually be constrained to the colour tool object, though this wasn't actually carried by the participant.

Having three sliders is cumbersome and it was suggested to the participant that he create a single 3D slider that would generate three outputs that could be connected up in the manner shown in Figure 6.7b. Deciding how this would be accomplished was the single most difficult task carried out by any of the experimental subjects, though having thoroughly investigated how a 1D slider object worked and with a little experimentation he was able to first build a 2D slider, and subsequently a 3D slider was a simple extension. The main step was realising that multiple *constrain between* functions (§5.5.5) could operate together to describe a completely bounded space. The design therefore required three orthogonally placed slider bars and a single slider button which was constrained between all three slider bars. The three between constraint filter functions generate three outputs and these are used as inputs to the colour tool function. The abstract representation of the 3D slider data flaw is shown in Figure 6.8. The immersive presentation of the final colour tool object is shown in Figure 6.9.

A second exploratory task was based upon one of the participant's suggestions. This was to create a simple world that investigated the properties of collision detection and dynamics in a different setting to that of the table

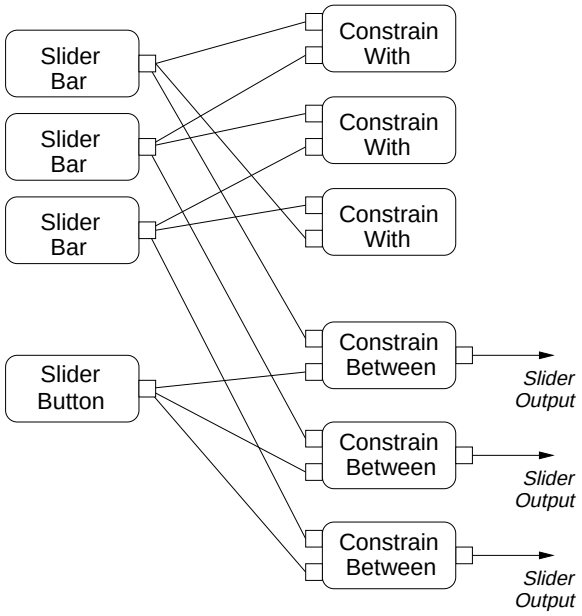


Figure 6.8: Abstract representation of the 3D slider

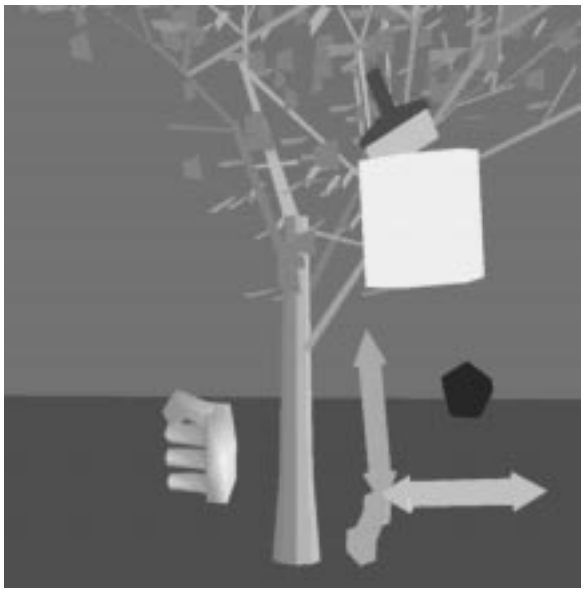


Figure 6.9: Immersive representation of the final colour tool object

tennis game. The world would contain a box with a ball inside that could be caught and thrown by the participant.

The components of this world were quite simple, the box, a ball identical to the one used in the table tennis application, and a technique for picking the ball similar to the one employed for picking the bat in the revised table tennis application (§6.3.2). However the construction was problematical and time constraints led to it's being left uncompleted.

The first problem was due to the fact that the containing box couldn't be a single object since collision detection in dVS is done on bounding boxes and not object surfaces. This would mean that the ball inside the cube would be continually colliding which would lead to a strange dynamics behaviour. Therefore the box had to be composed of six separate face objects. This led to the problem that such a face object didn't exist so a break had to be taken whilst the supervisor created a face object in 3D Studio and converted it into the required format. Once imported in to the world, the face objects had to be aligned and this was difficult since VEDA did not support any alignment tools. Once this was roughly accomplished the objects had to constrained to each other to form the complete cube.

The second problem was also due to the collision detection. In this case it was the fact that the constructed face objects were very thin and thus it was relatively easy for the ball to pass through a face since the collision test was based upon a bounding box overlap test at discrete time intervals[3]. A work around using *constrain between* functions was suggested, but lack of time prevented its implementation. This meant that the catch and throw behaviour was also left unimplemented.

Overall though this evaluation was successful since the construction of the colour tool was a task more difficult than any that was expected to be completed during the evaluation.

One of the main criticisms that this participant made during the sessions was that buttons gave no feedback when pressed[4]. After the final session the participant briefly sketched out solutions that used the colour and path following functions of VEDA.

[3]This was problem was encountered before during the construction of the table tennis application. Since it was due to the behaviour of dVS's collision actor there was no way to fix it directly, but for the table tennis it was worked around by joining the table top to the table legs so that the space below the table was effectively collidable

[4]Audio feedback was programmed into VEDA, but was not working due to a hardware fault during the evaluation.

6.3.3 VEDA Extension

Participant C undertook a more involved evaluation with the specific aim of extending VEDA with a more complete body model and testing the effect of the body model of the sense of presence within the environment. This involved two main tasks, writing an inverse kinematics filter to model the positions of body parts not directly tracked, and building a demonstration environment which could be used in an experiment.

Training of this participant followed similar path to the other two, but focussed on the data flow from the source and the various filters to do with manipulating object positions.

The kinematics filter was written in C, and the code linked into VEDA once a suitable interface to map the VEDA data inputs on to the function calls of the new module had been written. The mapping was a set of conversions from the VEDA position type, a four by four homogeneous matrix, to the filter's position type which used six floating point numbers, three for position and three euler angles for rotation. The filter took two inputs, the positions of the head and the hand, and generated ten outputs. These corresponded to two positions to model the free arm, the right shoulder and right elbow, and two sets of four positions to model the left and right legs each having the hip, knee, ankle and toe positions. The output data objects were presented in a known order, but to make things simple, the function representation was designed to have the input and output data objects in the correct positions about a model of a body.

Once these joint positions were available, a body was constructed by stretching limb objects between pairs of joint positions using stretch position filters (§5.5.6). Figure 6.10 illustrates this by showing the abstract data flow for the complete arm model. The first stage of the body construction was to test the stretching of each limb separately. Thus one limb was stretched between pairs of joint positions in turn. This quickly showed that the coordinate systems of VEDA and the new filter did not agree. Rectifying this involved the participant's re-writing the source code that mapped VEDA data streams onto the kinematics function inputs.

Once satisfied with the behaviour of the filter a complete body could be constructed by copying the existing limb a further seven times and reconnecting the inputs to the new objects. In addition a body object was attached to the head position. However it was found to be hard to inspect the overall behaviour of the body from an egocentric viewpoint and this led to the construction of a dummy body. The dummy body was not connected to the head and hand position in any way, but used the positions of two new meta objects as the head and hand. This meant that the body could be

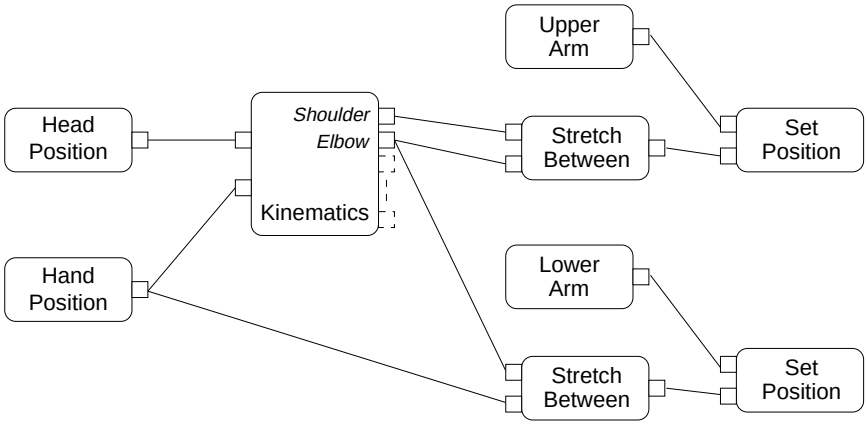


Figure 6.10: The abstract data flow model of the arm

manipulated much like a mannequin. Once the mannequin was completed and used to demonstrate the behaviour of the body, it was then animated by creating paths for the dummy head and hand to follow.

The remaining tasks for this participant were to model the geometry of the limbs and construct an experimental scenario. These were not completed at the time of writing and did not involve using VEDA itself.

Overall the main advantage that this participant gained from using VEDA was that there was no need to write a test harness for the kinematics filter in order to use it within a VE. Coding the body behaviours directly in C would have meant that any application requiring a body would have had to have been rewritten whereas with this approach the objects that defined the body could be imported into any environment.

This participant made similar comments to the others about using VEDA. Once over a few minor hurdles to do with behaviour of the system, he found it easy to make quick modifications to explore the behaviour of his filter.

6.3.4 Conclusions

The criteria for success were that participants would be able to understand and modify existing data flows, and create simple new data flows. Overall the user evaluation was very successful, with each participant performing reasonably complicated tasks.

Initially each participant found the view of the data flow network slightly confusing, but all of them quickly got used to this. Whether or not the 3D view was of intrinsic benefit to this task, as suggested in Section 5.2, was not explicitly determined by the evaluation, but two participants stated that

the 3D view helped them when tracing out specific data flow connections and also helped them to spatially separate the objects in to connected and interacting components.

It had also been expected that each participant would be able to make simple modifications to environments such as those listed in Section 6.1. Such modifications were easily understood conceptually and practically by the participants and they were all able to perform them. Furthermore all the studies involved phases where the participants had to create new data flows from unconnected components. None of them found this difficult, with one commenting that it made VE construction "similar to D.I.Y".

The main problems brought out by the experiment were the lack of geometry editing within VEDA and reliance on an external collision detection algorithm and consequent lack of control over behaviour. Both of these problems could be solved given an appropriate system.

Each participant was pleased with the tasks they managed to carry out and saw possibilities for customization that had not been anticipated. Many of the tasks undertaken were suggestions made whilst immersed within the application and most could be carried out immediately without any need to learn a new part of the toolkit.

6.4 VEDA As A Visual Programming Language

Data flow visual programming languages can be classified according to aspects of their design and their application domain [Hil92]. Hils uses the following criteria to distinguish between the design of VPLs and gives several examples for each:

- Pure data flow model. The VPL may have added control flow constructs which make it impure.
- Box-line representation. The standard representation has boxes, representing functions, connected by lines representing data flow, but this is not necessary.
- Iteration. The VPL may provide iteration in some way, by circular graphs or repeated execution for example.
- Procedural Abstraction. An entire graph can be condensed into a single node.

- Selector and Distributor. These two constructs are complimentary, the first selects which of two input values to pass to the output, the second selects to which of two outputs to send an input.
- Sequential Execution Construct. A VPL may supply some way to specify a sequence of functions to execute.
- Type Checking. There may or may not be type checking on the arcs between nodes.
- Higher-Order Functions. A VPL might allow functions themselves to be passed to other functions.
- Execution Modes. The data flow might be demand driven or data driven (§5.3).
- Level of Liveness. This refers to the mode in which the VPL operates. This range from static informative languages, to *live* languages which have a dynamic response to data or program changes.

Many of these criteria have been addressed during the description of the VEDA system, but we summarize them here.

VEDA is a **pure data flow** model VPL, in that it does not support control flow constructs like WHILE and CASE explicitly. It has a **box-line representation**, but as we have seen, this is extended by exploiting the richness of VEs as presentation environments, see Section 5.2. There is no direct support for **iteration**, though the data flow can be cyclic. There is support for iteration over collections of objects implicit in the operation of most of the filters themselves. There is **procedural abstraction** and we have seen that this provides a powerful method of hiding aspects of the virtual environment (Section 5.5.8). There is indirect support for **selector and distributor** functions using the simple filters of Section 5.5.3 and the use of multiple connections to input data objects. **Sequential execution** is supported using a *then* filter described in Section 5.5.3. **Type checking** is supported in construction, by connections not being allowed between different types of input/output pairs of different types (§5.5.2). **Higher-order functions** are not supported. As discussed in Section 5.3, the **execution mode** is data-driven rather than demand-driven. Finally the **level of liveness** is *live* since, unlike most data flow languages, VEDA is designed for immediate interactive use rather than static programming.

The representation techniques for VEDA are certainly novel compared to other 3D VPLs such as those described in Section 4.5. In VEDA the overriding design criteria is that the constructs of the language reflect the

positioning and attributes of the objects they affect in the presented application environment. This requires that the layout, at the top meta object layer in particular, reflect what is required from the application and can not be constrained by layout rules such as those in 3D cone trees [RMC91]. This free positioning of objects in the environment certainly restricts the legibility inherent in well regulated display such as cone-trees, but does allow the objects representing VEDA behaviours to be grouped with the objects they affect. The representation of individual nodes is also constrained by the objects required in application environments. There is little scope for object appearance such as hue, saturation and orientation to reflect attributes of the functions it represents, such as class hierarchy, state, and liveness. Such a scheme, see [FW94] for an example, would improve the legibility of the data flow but leave few choices for the design of the VE itself since the representation of a top level meta objects should reflect it's purpose within the application. For intermediate level meta objects down to function objects, the representation can reflect their function and the examples given illustrate this.

The interaction techniques for interacting with VEDA constructs are also novel, since they can use the full range of techniques possible in an immersive environment. These techniques can themselves be described by VEDA and thus there is room for experimentation with the best metaphors for using the VEDA tools in different application contexts.

The unique aspect of data flow languages for IVEs is the combined views of the presentation and definition levels of the data flow. This combination constrains the view of the definition level somewhat, but in combination with a live mode of operation it does provide an excellent prototyping environment.

Chapter 7

Conclusions

This thesis has presented and discussed the requirements for the Virtual Environment Dialogue Architecture (VEDA) system and evaluated its utility for creating virtual environments (VEs). This conclusion summarizes the work on VEDA, discusses wider implications of the approach taken and some possibilities for further work.

7.1 Contributions

The main contribution of this thesis have been the development of requirements for VEDA, the evolution of an approach to VE definition, the design of a VE model and the evaluation of VEDA. This section describes how these contributions have been presented.

7.1.1 Requirements for VEDA

Chapters 2 and 3 examined interaction metaphors with non-immersive virtual environment (NIVE) and immersive virtual environment (IVE) systems and presented a number of modalities inherent in interaction techniques and requirements for modification of interaction metaphor on a per user and per task basis. In particular Chapter 2 talked about a desktop interaction device called the Desktop Bat, though many of specific results about the device apply to any desktop system. Chapter 3 considered a navigation metaphor for IVEs called the Virtual Treadmill, and the effect on the sense of presence and task performance.

Chapter 3 also discussed the sense of presence in more detail and presented a model derived from experimental work. This hypothesised that the

matching the representation of self within the environment with the proprioceptive cues arising from interaction gives an enhanced sense of presence.

Enhancing this sense of presence leads naturally to using a combined environment within which modifications can occur in place using tools and metaphors consistent with that environment.

7.1.2 Approach

The study of NIVE and IVE interaction techniques gave some technical requirements for VEDA, but the superiority of an IVE over NIVE for space comprehension and simplicity of interaction led to VEDA's being designed to work in an IVE system primarily, though NIVE systems are not totally excluded. This solved a number of other problems with programming immersive systems that break down into two categories (§5.2):

- Describing some aspects of a VE with a standard description language is hard since the tasks are inherently three dimensional and easier to describe by demonstration or dietic gestures.
- Testing a IVE means repeatedly donning and removing the display hardware, to evaluate and then change object properties. For compiled systems this is compounded by the fact that this interactive loop maybe very slow.

VEDA solves this by allowing specification within the environment and since the execution mode is live, the changes made are active immediately. The gives a very fast prototyping environment, especially with the provision of virtual tools to aid the process. It uses the techniques of visual programming languages (§4.3), though it is unique in that field in that it is the only general language that is both presented with an IVE and used to describe the IVE itself (§4.5).

7.1.3 Model

VEDA uses a data flow model (§5.3) combined with an object hierarchy that provides encapsulation of behaviour detail. The components of the model come in three types: data objects which represent connection points for data streams, function objects that represent functions that manipulate data streams and meta objects that act as top level objects in the environment and also to encapsulate detail and provide higher level interfaces to the combined behaviour of a set of other objects. The data stream between objects are depicted using tubes that connect the two data object connection points.

The functions of the data flow come in three classes: device, tools and filters. Device functions represent sensors that measure external events and report these as data streams inside the environment. Tool functions represent actions that affect the properties of objects within the VE. Filter functions manipulate data streams to provide facilities such as logical combinations of trigger events, gesture recognition and collision detection. A standard set of functions derived from consideration of virtual reality scene description languages (§4.1)and programming libraries and (§4.2), is provided to enable the easy description of interesting environments.

The practical use of these functions is elaborated upon in Sections 5.6 and 5.7. The former describes some examples of the construction of environment objects with more complicated behaviour. And the latter describes a standard environment constructed with VEDA within which manipulation can take place. This provides some higher level tools for the participant to use and a standard set of interaction metaphors for editing.

7.1.4 Evaluation

The evaluation of VEDA considered its utility in creating and editing environment, both in terms of functionality and usability. Firstly VEDA's capabilities were checked against the requirements elaborated in Chapters 2 and 3. These were easily accomplished within VEDA and there were several extensions that became very simple to accommodate (§6.1).

Section 5.6 discussed some small examples of creating more complex behaviours out of atomic functions, but a more substantial application, virtual table tennis, was discussed in Section 6.2 in order to highlight how complicated data flow structures are handled. This indicated how new filters might be constructed and easily inserted into the application and how encapsulation of behaviour inside meta objects make the data flow more comprehensible.

The main part of the evaluation was three in depth trials where participants were invited to modify and create environments, see Section 6.3. The participants were able to understand the VEDA model and despite some initials problems with the definition views, they able to interact and understand immersive representations of the data flow. In subsequent experimentation and exploration sessions, the participants undertook and successfully completed tasks that were more complicated than expected. The sessions also reiterated the breadth of possibilities for interaction and the possible benefits for spatial comprehension that IVEs give.

Finally the evaluation classified VEDA as a visual programming language to illustrate the breadth of current languages and the choices that were made in VEDA's design.

7.2 Discussion

The approach to environment creation that VEDA enshrines is radically different from most other VE toolkits in existence at the moment. Some toolkits have exploited the immersive properties of an environment for geometry editing, but no other uses an immersive system to define and modify behaviour of the objects. In some ways the approach VEDA takes may be more appropriate for programming IVEs since it doesn't involve the participant's having to leave the VE and make changes in a radically different view before returning to the VE. It is intrinsically a naturalistic metaphor for environment editing since it is analogous to modifying a real world machine by taking it apart, tweaking the components and then setting it running again. This analogy has been exploited to some extent with the toolbelt and toolbox object hierarchies within a workshop environment.

The data flow model underlying VEDA provides a powerful description technique that was easily learned by participants. A very similar model with similar filters types forms the basis of the emerging VRML2.0 specification [VRM96]. Broadly speaking, VRML2.0 specifies a similar object hierarchy to that of VRML1.0 (§4.1), with the addition of script nodes that can read and write the field values of objects. Scripts and fields are connected together using *ROUTES* which correspond to data flow connections in VEDA. However VRML2.0 has no intrinsic visualization of the data flow, and only the VEDA equivalent of top level meta objects are ever shown to the user. The VRML2.0 model of data flow and execution is more sophisticated than that of VEDA, in that portions of the data flow that drive objects that are not within view can be marked so they won't be evaluated. This hybrid data and demand driven data flow is of potential benefit once the simulation time for a world becomes such that it is impossible to propagate the data flow to completion on each frame. Such considerations have not arisen in current VEDA worlds which have over a thousand objects and hundreds of data connections.

The immersive arrangement of the data flow has immediate benefits for constructing larger scale environments. Being able to encapsulate functions and meta objects within the object they define encourages an object-orientated approach with hierarchies of components that can be copied and instanced. The approach is obviously different from a programming language that allows inheritance and sub-classing since behaviour components can only be added and removed but encapsulation allows a form of data hiding and abstraction.

The main drawbacks with using VEDA were to do with understanding the data flow layouts and interaction techniques of the environment. Any prob-

lems that were encountered with comprehension of the environment were quickly overcome through exploration of the environment and revealing and hiding relevant parts of the hierarchy. The process of becoming familiar with the model of the environment's behaviour can be eased somewhat by making the techniques more naturalistic and this was a guiding principle when designing VEDA's tools. Of course VEDA provides a good basis from which to explore what these immersive techniques might be and many suggestions were made by the participants whilst immersed in the environments. In fact since the participants were present in the application environment at the time they made these suggestions, it could be argued that the techniques they proposed were more suited to the overall metaphor for the environment than others that would have been suggested when outside the environment. For this reason it is hoped that through continued exploration with VEDA, different approaches to interaction techniques may be discovered.

7.3 Future Work

There are many avenues for future investigation with a system such as VEDA to provide such things as collaborative modification, distributed interfaces and more complex behaviour.

Currently the use of VEDA is limited by two main issues: the single platform implementation and single user environment. The platform dependence restricts the use of VEDA to those systems running a particular version of Division's dVS, a proprietary standard. There are several limitations inherent in this approach, mainly due to the level of abstraction and consequent loss of efficiency at which VEDA interfaces with dVS (§5.8). Given the high level at which VEDA interacts with the host operating system, it would be interesting to investigate not only whether porting VEDA to another system would be feasible, but whether or not a VEDA-like system could be implemented in a VE scripting language such as Dive-TCL (§4.1) or VRML2.0. This would remove any browser dependency at all, and would allow an authors viewing, say a VRML2.0 world, to simply load in VEDA when required, visualize the data flow with the included objects, make the required changes, and then remove VEDA when finished.

The other current restriction is that VEDA is a single user system. That is not to say it cannot be used in a shared environment, but participants are restricted to simply viewing the other's data flow hierarchy and can not interact with it. Also since VEDA only propagates visual and audio information into the dVS object database, there is no way for two functions on different hosts to set up a data flow connection. The connection of streams

between different systems would provide a powerful model of execution, but would introduce new complexity both at the database level and application level. Some issues to consider are how to efficiently map multiple streams onto a message passing mechanism, and perhaps more interestingly the issues of privacy of aspects of the data flow. In a distributed application there would certainly be a need to classify parts of the hierarchy as *private* and *public* so that one participant could not affect, say, something in the other's *participant hierarchy*, but they could collaborate on building a shared virtual environment.

Many aspects of the single-machine and single-user VEDA could undoubtably be improved. As was seen in Section 6.3 there are still general usability issues to do with IVEs. VEDA does however provide a coherent framework to investigate those very techniques.

Other schemes for representing the data flow should be investigated, and perhaps some of the informal rules for layout developed by VEDA's users should be made more concrete. One such informal rules simply dictates that the input and output objects of a meta object or function object should be evenly spaced out below the object. A more concrete rule might enforce a cone-tree [RMC91] type layout on the visible portions of an object's hierarchy.

At a deeper level, VEDA's data flow model is only one of several possible choices and from the classification of VEDA as a visual programming language (§6.4), it can be seen that there are many variations the current scheme that could be investigated. Existing work on 2D data flow languages (§4.4), illustrates some of these variations. One variation that should be useful is the programming by demonstration model (§4.3.2).

Finally, given that one of the main motivations of VEDA to remove the edit-compile-experience cycle from IVE construction, an immediate plan is to integrate VEDA with a geometry editing environment so that the whole process of constructing an environment can take place in the space that the environment will eventually fill.

Appendix A

VEDA Functions

Sensor Functions

Function Name	*Purpose*
trigger	Reports the state of the trigger button of the 3D mouse
left	Reports the state of the left button of the 3D mouse
middle	Reports the state of the middle button of the 3D mouse
right	Reports the state of the right button of the 3D mouse
bottom	Reports the state of the bottom button of the 3D mouse
absoluteHand	Reports the position of the hand
absoluteHead	Reports the position of the hand
relativeHand	Reports the relative position of the hand with respect to the head
relativeHead	Reports the position of the body in the world as the projection of the head position onto the floor
time	Reports the current clock time

Basic Data Flow Functions

Function Name	*Purpose*
copy	Copies an object hierarchy
select	Selects an object hierarchy and adds it to a selection set
delete	Deletes an object hierarchy
hide	Hides an object hierarchy
reveal	Reveals an object hierarchy
encapsulate	Encapsulates an object hierarchy within another object
deencapsulate	Deencapsulate an object hierarchy from a hierarchy
pick	Picks a set of objects, i.e. temporarily constraints them to the hand object
save	Saves a selected set of objects or the complete world if the selection set is empty

Logic and Gesture Functions

Function Name	*Purpose*
and	Logical and of two data streams
not	Logical negation of a data streams
or	Logical or of two data streams
equal	Logical equivalence of two data streams
doubleClick	Recognition of two true values in a specified time
then	Recognition of first one then the second input becoming true within a specified time
delay	Delays a data stream by a specified time
gate	Allows data streams to be suspended
toggle	Output value switches each time the input becomes true and then false
gestureRecognise	Recognize a gesture from a position stream using a specified feature based gesture
gestureTrain	Train a feature based gesture from a position stream
neuralNet	Train and recognize a neural net based gesture

Object Property Functions

Function Name	*Purpose*
specificCollide	Reports collision between two specified objects
generalCollide	Reports collisions with a specified object
constrainTo	Constrains one object to another object
constrainBetween	Constrains one object to lie between the endpoints of the X-axis of another object
constrainAlong	Constrains one object to lie along the X-axis of another object
dynamics	Controls the simulation of gravity and collision acting upon an object
colour	Applies a colour to an object
scale	Applies a scale to an object
move	Moves an object in a specified direction
makePath	Create an animation path
followPath	Moves an object along an animation path

Position Functions

Function Name	*Purpose*
getPosition	Returns the position of an object
setPosition	Sets the position of an object
getRelative	Get the relative transformation between two positions
setRelative	Applies the relative transformation to positions to obtain a composite position
invertPosition	Inverts a position
stretchPosition	Creates a position that would stretch a unit long object between two positions

Miscellaneous Functions

Function Name	*Purpose*
kinematics	Generates body part positions from position of a hand and head
attract	Simulates a force being applied to an object
tubeDispenser	Creates new tubes for data flow connection
multimeter	Debugging function
opponent	Opponent simulation component of the table tennis application

Bibliography

[aMMB89] A. L. Ambler an M. M. Burnett. Influence of visual technology on the evolution of language environments. *IEEE Computer*, 6(2):9–22, October 1989.

[BBH+90] C. Blanchard, S. Burgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel. Reality built for two: A virtual reality tool. *Computer Graphics*, 24(2):35–36, March 1990.

[BBL93] T. Baudel and M Beaudouin-Lafon. Charade: Remote control of objects using free-hand gestures. *Communications of the ACM*, 36(7):28–35, 1993.

[BC94] G. Burdea and P. Coiffet. *Virtual Reality Technology.* Wiley Interscience, 1994.

[BDHO92] Jeff Butterworth, Andrew Davidson, Stephen Hence, and T.Marc Olano. 3DM: A three dimensional modeller using a head mounted display. *Computer Graphics. Special Issue 1992 Symposium on Interactive 3D Graphics*, pages 135–138, 1992.

[BF95] W. Barfield and T.A. Furness III, editors. *Virtual Environments and Advanced Interface Design.* Oxford University Press, 1995.

[BH93] Monica Bordegoni and Matthia Hemmje. A dynamic gesture language and graphical feedback for interaction in a 3D user interface. *Computer Graphics Forum*, 12(3):C1–C11, 1993. Proceedings of EUROGRAPHICS'93.

[BH95] W. Barfield and C. Hendrix. The effect of update rate on the sense of presence within virtual environments. *Virtual Reality: The Journal of the Virtual Reality Society*, 1(1):3–16, 1995.

[BHV92] K. Böhm, W. Hübner, and K. Väänänen. Given: gesture driven interactions in virtual environments. A toolkit approach to 3D

interactions. In *Informatique 92: Interface to Real and Virtual Worlds, Montpellier France, 23-27 March*, pages 243–254, 1992.

[Bie86] E.A. Bier. Skitters and jacks: Interactive 3D positioning tools. In F. Crone and S. M. Pizer, editors, *Proc. 1986 ACM Workshop on Interactive 3D Graphics, Chapel Hill, NC, October 22-24*, pages 183–196. ACM Press, 1986.

[Bie90] E.A. Bier. Snap-dragging in three dimensions. *Computer Graphics*, 24(2):193–203, 1990. Special Issue on Symposium on Interactive 3D Graphics.

[BM86] W. Buxton and B.A. Myers. A study in two-handed input. In *CHI'86 Proceedings*. ACM Press, New York, 1986.

[BMB86] N.I. Badler, K.H. Manoochechri, and D. Baraff. Multi-dimensional input techniques and articulated figure positioning by multiple contraints. In F. Crone and S. M. Pizer, editors, *Proc. 1986 ACM Workshop on Interactive 3D Graphics, Chapel Hill, NC, October 22-24*, pages 151–169. ACM Press, 1986.

[Bor81] A. Borning. The programming language aspects of thinglab, a constraint-oriented simulation laboratory. *ACM Trans. Programming Languages and Systems*, 3(4):353–387, 1981.

[Bor86a] A. Borning. Defining constraints graphically. In *Proceedings of CHI'86*, pages 137–143. ACM Press, 1986.

[Bor86b] A. Borning. Graphically defining new building blocks in ThingLab. *Human Computer Interaction*, 2:269–295, 1986.

[BPP95] Gavin Bell, Anthony Parisi, and Mark Pesce. The virtual reality modeling language, version 1.0 specification. Web document, available at http://vrml.wired.com/vrml.tech/vrml10-3.html, May 26 1995.

[Bri93] Louis M. Brill. Facing interface issues. *Computer Graphics World*, 15(4), April 93.

[Bro88a] M.H. Brown. Exploring algorithms using balsa-ii. *Computer*, pages 14–36, May 1988.

[Bro88b] M.H. Brown. Perspectives on algorithm animation. In *Proc. CHI'88: Human Factors in Computing Systems*, pages 33–38. ACM Press, 1988.

[BS84] M.H. Brown and R. Sedgewick. A system for algorithm animation. *Computer Graphics (Proceedings of SIGGRAPH'94)*, 18(3):177–186, 1984.

[BS85] M.H. Brown and R. Sedgewick. Techiques for algorithm animation. *IEEE Software*, pages 28–39, January 1985.

[BSZS95] W. Barfield, T. Sheridan, D. Zeltzer, and M. Slater. Presence and performance within virtual environments. In W. Barfield and T. Sheridan, editors, *Virtual Environments and Advanced Interface Design*. Oxford University Press, 1995.

[CFD+89] Upson C., Faulhaber Jr.T., Kamins. D, Laidlaw D., Schlegel D., Vroom J., Gurwitz R., and A. van Dam. The application visualization system: A computational environment for scientific visualizaion. *IEEE CG & A*, pages 30–42, September 1989.

[CH93] C. Carlsson and O. Hagsand. Dive - a platform for multi-user virtual environments. *Computers and Graphics*, 17(6), 1993.

[Cha87] S.-K. Chang. Visual languages: A tutorial and survey. *IEEE Software*, pages 29–39, January 1987.

[Cha90] S.-K. Chang, editor. *Principles of Visual Programming Systems*. Prentice Hall, New York, 1990.

[CHB+89] J.C. Chung, M.R. Harris, F.P. Brooks, H. Fuchs, M.T. Kelley, J. Hughes, M. Ouh-Young, C. Cheung, R.L. Holloway, and M. Pique. Exploring virtual worlds with head-mounted displays. *SPIE Vol. 1083 Three-Dimensional Visualization and Display Technologies*, pages 42–52, 1989.

[CMS88] M. Chen, S.J. Mountford, and A. Sellen. A study in interactive 3-D rotation using 2-D devices. *Computer Graphics*, 22(4):121–129, August 1988.

[CP88] P.T. Cox and T. Pietrzykowski. Using a pictorial representation to combine dataflow and object orientation in a language independent programming mechanism. In *Proceedings International Computer Conference*, pages 695–704. IEEE, 1988.

[CRP95] A. Colebourne, T. Rodden, and K. Palfreyman. VR-MOG: a toolkit for building shared virtual worlds. In M. Slater, editor,

Proceedings Conference of the FIVE working group (QMW University of London, UK, 18-19 December 1995). Queen Mary and Westfield College, University of London, Mile End Road, London, E1 4NS, UK, 1995. ESPRIT Working Group 9122.

[CW92] D. Chapman and C. Ware. Manipulating the future: Predictor based feedback for velocity control invirtual environment navigation. *Computer Graphics. Special Issue 1992 Symposium on Interactive 3D Graphics*, pages 63–66, 1992.

[Cyp91] A. Cypher. Eager: Progamming repetitive tasks by example. In *Proc. CHI '91*, pages 33–39, New Orleans, Louisiana, April 1991. ACM Press, New York.

[Dee92] Michael Deering. High resolution virtual reality. *Computer Graphics*, 26(2):195–202, July 1992.

[DGB+79] R. Dilts, J. Grinder, J. Bandler, L. DeLozier, and L. Cameran-Bandler. *Neuro-Linguistic Programming I.* Meta Publication, 1979.

[Div92] Division Ltd., 19 Apex Court, Woodlands, Almondsbury,Bristol,NS12 4JT, U.K. *Provision dVS manual*, 1992. version 0.2.

[Div94a] Division Ltd., 19 Apex Court, Woodlands, Almondsbury,Bristol,NS12 4JT, U.K. *dVISE User Guide*, 1994.

[Div94b] Division Ltd., 19 Apex Court, Woodlands, Almondsbury,Bristol,NS12 4JT, U.K. *dVS Technical Overview*, 1994. Version 2.0.4.

[Div94c] Division Ltd., 19 Apex Court, Woodlands, Almondsbury,Bristol,NS12 4JT, U.K. *xDVISE User Guide*, 1994.

[DRP+92] N.I. Durlach, A. Rigopulos, X.D. Pang, E.m. Wenzel, W.S. Woods, A. Kulkarni, and H.S. Colburn. On the externalization of auditory images. *Presence: Teleoperators and Virtual Environments*, 1(2):251–257, 1992.

[Dui88] R.A. Duisberg. Animation using temporal constraints: An overview of the animus system. *Human Computer Interaction*, 3(3):275–307, 1987/88.

[Ell91] S.R. Ellis. Nature and origins of virtual environments: A bibliographical essay. *Computing Systems in Engineering*, 2(4):321–347, 1991.

[FG84] W. Finzer and L. Gould. Programming by Rehearsal. *BYTE*, 9(6):187–210, June 1984.

[FH93] S. Sidney Fels and Geoffrey E. Hinton. Glove-talk: A neural network based interface between a data-glove and a speech synthesizer. *IEEE Transactions on Neural Networks*, 4(1):2–8, January 1993.

[FH95] Emmanuel Frécon and Olof Hagsand. *The Dive/Tcl Behaviour Interface.* Swedish Institute of Computer Science, Stockholm, November 23 1995. Available from http://www.sics.se/dive/manual/tcl-behaviour.html.

[Fis90] S.S. Fisher. Virtual interface environments. In Brenda Laurel, editor, *The Art of Human-Computer Interface Design.* Addison Wesley, 1990.

[FMHR86] S.S. Fisher, M. McGreevy, J. Humphries, and W. Robinett. Virtual environment display system. *Proc. 1986 ACM Workshop on Interactive 3D Graphics, Chapel Hill, NC, October 22-24*, pages 77–87, 1986.

[Fol86] J.D. Foley. Dynamic process visualization. *IEEE CG & A*, 6(3):16–25, 1986.

[Fol87] James D. Foley. Interfaces for advanced computing. *Scientific American*, pages 83–90, October 1987.

[FW94] G. Franck and C. Ware. Representin nodes and arcs in 3D networks. In *Proc. 1994 IEEE Symposium Visual Languages*, pages 189–190, 1994.

[FWC84] James D. Foley, Victor L. Wallace, and Peggy Chan. The human factors of computer graphics interaction techniques. *IEEE Computer Graphics and Applications*, pages 13–48, November 1984.

[GB93] Enrico Gobbetti and Jean-Francis Balaguer. VB2 an architecture for interaction in synthetic worlds. In *Proceedings User Interface and Software Technology (UIST'93)*. ACM Press, New York, 1993.

[GB95] Enrico Gobbetti and Jean-Francis Balaguer. An integrated environment to visually construct 3d animations. In *SIGGRAPH 95 (Los Angeles, CA, August 6-11, 1995)*, Computer Graphics Proceedings, Annual Conference Series, pages 395–398. ACM SIGGRAPH, 1995.

[Gli87] E.P. Glinert. Out of flatland: Towards 3-d visual programming. In *Proc. of the 2nd Fall Joint Computer Conference*, pages 292–299. IEEE Computer Society Press, 1987.

[GMD90] E. P. Glinert, Kopache M.E., and McIntyre D.W. Exploring the general-purpose visual alternative. *J. Visual Languages and Computing*, 1(1):3–40, 1990.

[Gol91] E. J. Goline. Tool review: Prograph 2.0 form TGS systems. *J. Visual Languages and Computing*, 2(2):189–194, 1991.

[GT84] E. P. Glinert and S. Tanimoto. Pict: An interactive graphical programming language. *IEEE Computer*, 7(25), November 1984.

[Hae88] P.E. Haeberli. ConMan: A visual programming language for interactve graphics. *Computer Graphics (Proceedings SIGGRAPH 88)*, 22(4):103–111, 1988.

[Har95] Jan Hardenbergh. Course 12, VRML: Using 3D to surf the web. In *SIGGRAPH 95, Course Notes*. ACM SIGGRAPH, 1995.

[HD92] R.M. Held and N.I. Durlach. Telepresence. *Presence: Teleoperators and Virtual Environments*, 1(1):109–112, 1992.

[Hee92] Carrie Heeter. Being there: The subjective experience of presence. *Presence: Teleoperators and Virtual Environments*, 1(2):262–271, 1992.

[Her80] C.F. Herot. Spatial management of data. *ACM Trans. on Database Systems*, 5(4):493–513, 1980.

[Hil92] D.D. Hils. Visual languages and computing survey: Data flow visual programming languages. *J. Visual Languages and Computing*, 3(1):69–101, 1992.

[HKP91] J. Hertz, A. Krough, and R.G. Palmer. *Introduction to the Theory on Neural Computation*. Addison-Wesley, 1991.

[IH87] T. Ichikawa and M. Hirakawa. Visual programming - toward realization of user-friendly programming environments. In *Proc. of the 2nd Fall Joint Computer Conference*, pages 129–136. IEEE Computer Society Press, 1987.

[IWC+88] D. Ingalls, S. Wallace, Y.-Y. Chow, F. Ludolph, and K. Doyle. Fabrik – a visual programming environment. In *Proc. OOPSLA '88*, pages 176–190, San Diego, 1988.

[Kal93a] Roy S. Kalawsky. *The Science of Virtual Reality and Virtual Environments.* Addison-Wesley, 1993.

[Kal93b] R.S. Kalawsky. The science and engineering of virtual reality. In *Virtual Reality International 93, Proceedings of the third annual conference on Virtual Reality, London, April 1993.* Meckler, 1993. Not contained in published proceedings.

[KCM90] T. D. Kimura, J. W. Choi, and J. M. Mack. Show and tell: A visual programming language. In E. P. Glinert, editor, *Visual Programming Environments: Paradigms and Systems*, pages 397–404. IEEE Computer Society Press, 1990.

[Ken80] A. Kendon. Gesticulations and speech: Two aspects of the process of utterance. In R.A. Key, editor, *The Relation between Verbal and Non-Verbal Communication.* Mouton, The Hague, 1980.

[Kru90] Myron W. Krueger. Videoplace and the interface of the future. In Brenda Laurel, editor, *The Art of Human-Computer Interface Design.* Addison Wesley, 1990.

[Kru91] Myron W. Krueger. *Artificial Reality II.* Addison Wesley, second edition, 1991. ISBN 0-201-52260-8.

[Lan91] J.A. Landay. Tools review: Serius - a visual programming environment. *J. Visual Languages and Computing*, 2(3):297–303, 1991.

[LCI+88] F. Ludolph, Y.-Y. Chow, D. Ingalls, S. Wallace, and K. Doyle. The Fabrik programming environment. In *Proc. 1988 IEEE Workshop Visual Languages*, pages 222–230. IEEE Computer Society Press, 1988.

[Lip91] J.S. Lipscomb. A trainable pattern recognizer. *Pattern Recognition*, pages 895–907, 1991.

[LKL91] J.Bryan Lewis, Lawrence Koved, and Daniel T. Ling. Dialogue structure for virtual worlds. In *CHI'91 Proceedings*. ACM Press, New York, 1991.

[Loo92a] J.M. Loomis. Distal attribution and presence. *Presence: Teleoperators and Virtual Environments*, 1(1):113–119, 1992.

[Loo92b] J.M. Loomis. Presence and distal attribution: Phenomenology, determinants, and assessment. In *SPIE Vol. 1666 Human Vision, Visual Processing, and Digital Display III*, pages 590–595, 1992.

[Lyt95] W. Lytle. Vpla: Visual programming language for animation. Technical Sketch, SIGGRAPH95, 1995.

[MAB92] K. Meyer, H.L. Applewhite, and F.A. Biocca. A survey of position trackers. *Presence: Teleoperators and Virtual Environments*, 1(2):173–200, 1992.

[McK92] M. McKenna. Interactive viewpoint control and three-dimensional operations. *Computer Graphics*, pages 53–56, 1992. Special Issue on Symposium on Interactive 3D Graphics.

[MCR90] J.D. Mackinlay, S.K. Card, and G.G. Robertson. Rapid controlled movement through a virtual 3D workspace. *Computer Graphics*, 24(4):171–176, August 1990.

[MH85] M. Moriconi and D.F. Hare. Visualizing program designs through PEGASYS. *IEEE Computer*, 18(8):72–85, 1985.

[Min84] M.R. Minsky. Manipulating simulated objects with real-world gestures using a force and position sensitive screen. *Computer Graphics (SIGGRAPH Proceedings)*, 18(3):195–203, 1984.

[Min95] Mark R. Mine. Virtual environment interaction techniques. In *Course 8: Programming Virtual Worlds*, SIGGRAPH 95, Course Notes. ACM SIGGRAPH, 1995.

[Moh88] T.G. Moher. Provide: A process visualization and debugging environment. *IEEE Trans. Software Engineering*, SE-14(6):849–857, 1988.

[MT91] K. Murakami and H. Taguchi. Gesture recognition using recurrent neural networks. In *CHI'91 Proceedings*. ACM Press, New York, 1991.

[Mye86] B.A. Myers. Visual programming, programming by example and program visualization: A taxonomy. In *Proceedings of CHI'86*. ACM Press, 1986.

[Mye87] B.A. Myers. Creating interaction techniques by demonstration. *IEEE CG & A*, pages 51–60, September 1987.

[Mye90a] B. A. Myers. Creating user interface using programming by example, visual programming and constraints. *ACM Trans. Programming Languages and Systems*, 12(2):143–177, 1990.

[Mye90b] B. A. Myers. Taxonomies of visual programming and program visualization. *J. Visual Languages and Computing*, 1(1):97–123, 1990.

[Naj94] M. Najork. *Programming in Three Dimensions*. PhD thesis, University of Illinois at Urbana-Chapaign, 1994.

[NK91] M. Najork and S. Kaplan. The cube language. In *Proc. 1991 IEEE Workshop Visual Languages*, pages 218–224, Kobe, Japan, 1991. IEEE Computer Society Press.

[NK92] M. Najork and S. Kaplan. A prototype implementation of the CUBE language. In *Proc. 1992 IEEE Workshop Visual Languages*, pages 270–273. IEEE Computer Society Press, 1992.

[NO86] G.M. Nielson and D.R. Olsen Jr. Direct manipulation techniques for 3D objects using 2D locator devices. In F. Crone and S. M. Pizer, editors, *Proc. 1986 ACM Workshop on Interactive 3D Graphics, Chapel Hill, NC, October 22-24*, pages 175–182. ACM Press, 1986.

[Ous94] John K. Ousterhout. *Tcl and the Tk toolkit*. Addison-Wesley, 1994.

[Pat92] Robert Patterson. Human stereopsis. *Human Factors*, 34(6):669–692, 1992.

[Pat94] H. Patel. ??? Master's thesis, Department of Computer Science, Queen Mary and Westfield College, University of London, 1994.

[PBS93] B.A. Price, M.B. Baecker, and I.A. Small. A principled taxonomy of software visualization. *J. Visual Languages and Computing*, 4(3):211–266, 1993.

[PN83] M.C. Pong and N. Ng. Pigs - a system for programming with interactive graphical support. *Software - Practice and Experience*, 13(9):847–855, 1983.

[Pol] Polhemus, Inc. Fastrak^R. One Hercules Drive,P.O. Box 560,Colchester, VT 05446,USA.

[PTVM92] J. Poswig, K. Teves, G. Vrankar, and C. Moraga. Visavis - contributions to practice and theory of highly interactive visual languages. In *Proc. 1992 IEEE Workshop Visual Languages*, pages 155–162, Seattle, WA, 1992.

[PVM94] J. Poswig, G. Vrankar, and C. Moraga. Visavis: a higher-order functional visual programminmg languages. *J. Visual Languages and Computing*, 5(1):83–111, 1994.

[RCM89] G.G. Robertson, S.K. Card, and J.D. Mackinlay. The cognitive coprocessor architecture for interactive user interfaces. In *Proceedings UIST'89*, pages 10–18, 1989.

[Rei85] S.P. Reiss. Pecan: Program development systems that support multiple views. *IEEE Trans. Software Engineering*, SE-11(3):276–285, 1985.

[Rei87] S.P. Reiss. Working in the GARDEN environment fo conceptual programming. *IEEE Software*, 4(6):16, 27 1987.

[RGR89] S.P. Reiss, E.J. Golin, and R.V. Rubin. Using GELO to visualize software systems. In *Proc. 2nd Annual Symp. on User Interface and Software Technology (UIST'89)*, pages 147–157. ACM Press, New York, 1989.

[Rhe91] Howard Rheingold. *Virtual Reality*. Secker & Warburg, London, 1991.

[RL84] N. Roussopoloulos and D. Leifker. An introduction to psql: A pictorial structured query language. In *Proc. 1984 IEEE Workshop Visual Languages*, pages 77–87, 84.

[RMC91] G.G. Robertson, J.D. Mackinlay, and S.K. Card. Cone trees: animated 3D visualizations of hierarchical information. In *Proceedings of the ACM SIGCHI 91 Conference on Human Factoirs in Computing*, pages 189–194, 1991.

[Rob92] W. Robinett. Synthetic experience: A proposed taxonomy. *Presence: Teleoperators and Virtual Environments*, 1(2):229–247, 1992.

[RR92] W. Robbinett and J.P Rolland. A computational model for the stereoscopic optics of a head mounted display. *Presence: Teleoperators and Virtual Environments*, 1(1), 1992.

[Rub91] Dean Rubine. Specifying gesture by example. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):329–337, 1991.

[RW91] J.R. Rasure and C.S. Williams. An integrated data flow visual language and software development environment. *J. Visual Languages and Computing*, 2(3):217–246, 1991.

[SAU94] M. Slater, C. Alberto, and M. Usoh. In the building or through the window? an experimental comparison of immersive and non-immersive walkthroughs. In *Virtual Reality Environments in Architecture, Leeds 2-3rd November*. Computer Graphics Society, 1994.

[SD91] M. Slater and A. Davidson. Liberation from flatland: 3D interaction based on the desktop bat. In *EUROGRAPHICS '91*, pages 209–221. Elsevier Science Publishers B.V. (North-Holland), 1991.

[She92a] T.B. Sheridan. Defining our terms. *Presence: Teleoperators and Virtual Environments*, 1(2):272–274, 1992.

[She92b] T.B. Sheridan. Musings on telepresence and virtual presence. *Presence: Teleoperators and Virtual Environments*, 1(1):120–126, 1992.

[She93] W.R. Sherman. Integrating virtual environments into the dataflow paradigm. In *Fourth Eurographics Workshop on ViSC, Abingdon, UK*, April 1993.

[Shn83] B. Shneiderman. Direct manipulation: A step beyond programming languages. *IEEE Computer*, pages 57–69, August 1983.

[Shu85] N. C. Shu. FORMAL: A forms-oriented and visual-directed application system. *IEEE Computer*, 18(8):38–49, 1985.

[Shu86] N.C. Shu. Visual programming langauges. a perspective and dimensional analysis. In S.-K. Chang, T. Ichikawa, and P. A. Ligomenides, editors, *Visual Languages*, pages 11–34. Plenum Press, New York, 1986.

[Shu88] N. C. Shu. *Visual Programming*. Van Nostrand Reinhold, New York, 1988.

[SIG89] SIGGRAPH'89. Virtual environments and interactivity. *Computer Graphics*, 23(5):7–38, 1989. Panel Proceddings.

[SIG90] SIGGRAPH'90. Hip, hype and hope-the three faces of virtual worlds. *Computer Graphics*, pages 10.1–10.29, 1990. Panel Proceddings.

[Sil] Silicon Graphics, Inc. Webspace. Available from http://webspace.sgi.com/.

[SLGS92] Chris Shaw, Jiandon Liang, Mark Green, and Yunqi Sun. The decoupled simulation model for virtual reality systems. In *Proc CHI'92, May 3-7, 1992*, pages 321–328, 1992.

[Smi77] D. C. Smith. *Pygmailion - A Computer Program to Model and Stimulate Creative Thought*. Birkhauser, Basel, 1977.

[SN93] D. Song and M. Norman. Nonlinear interactive motion control techniques for virtual space navigation. In *Proceedings of IEEE 1993 Virtual Reality Annual International Symposium, VRAIS '93, Piscataway, NJ.*, pages 111–117, 1993.

[SP92a] J.T. Statsko and C. Patterson. Understanding and charaterizing software visualization systems. In *Proc. 1992 IEEE Workshop Visual Languages*, pages 3–10, Seattle, WA, 1992.

[SP92b] R. Stiles and M. Pontecorvo. Lingua graphica: A visual language for virtual environments. In *Proc. 1992 IEEE Workshop Visual Languages*, pages 225–227. IEEE Computer Society Press, 1992.

[Spa] Spacetec Imc Corporation. Spaceball^R 2003^{TM}. 600 Suffolk Street, Lowell, MA 01854-3629, USA.

[SS94] A. Steed and M. Slater. A user-defined virtual environment dialogue architecture. In *Proceedings of VRST '94 - Virtual Reality Software and Technology*, pages 87–96. World Scientific Publishing Company, 1994.

[SS95] A. Steed and M. Slater. 3d interaction with the desktop bat. *Computer Graphics Forum*, 14(2), 1995.

[SS96] A. Steed and M. Slater. A dataflow representation for defining behaviours within virtual environments. In *Proceedings of VRAIS'96*, pages 163–167. IEEE, IEEE Computer Society Press, 1996.

[SSU93] M. Slater, A. Steed, and M. Usoh. The virtual treadmill: A naturalistic metaphor for navigation in immersive virtual environments. In M. Göbel, editor, *First Eurographics Workshop on Virtual Environments, Polytechnical University of Catalonia, September 7*, pages 71–83, 1993.

[SSU94] A .Steed, M. Slater, and M. Usoh. Presence in immersive virtual environments. In *Proceedings of the 1st UK Virtual Reality Special Interest Group Conference*, 1994.

[Ste92] J. Steur. Defining virtual reality: Dimensions determining telepresence. *Journal of Communication*, 42(4), 1992.

[Stu92] David J. Sturman. *Whole-Hand Input.* PhD thesis, Massachusetts Institute of Technology, February 1992.

[SU93] M. Slater and M. Usoh. The influence of a virtual body on presence in immersive virtual environments. In *Virtual Reality International 93, Proceedings of the third annual conference on Virtual Reality, London, April 1993*, pages 34–42. Meckler, April 1993.

[SU94a] M. Slater and M. Usoh. Body centred interaction in immersive virtual environments. In M. Magnenat-Thalmann and D. Thalmann, editors, *Virtual Reality and Artificial Life.* John Wiley, 1994.

[SU94b] M. Slater and M. Usoh. Representation systems, perceptual position and presence in virtual environments. *Presence: Teleoperators and Virtual Environments*, 2(3), 1994.

[SU92] Mel Slater and Martin Usoh. An experimental exploration of presence. Department of Computer Science, Queen Mary and Westfield College, University of London, Report, 92.

[SU93] M. Slater and M. Usoh. Presence in virtual environments. In *Proceedings of VRAIS'93*, pages 90–96. IEEE, September 93.

[SUS94a] M. Slater, M. Usoh, and A. Steed. Depth of presence in virtual environments. *Presence: Teleoperators and Virtual Environments*, 3(2), 1994.

[SUS94b] M. Slater, M. Usoh, and A. Steed. Steps and ladders in virtual reality. In *Proceedings of VRST '94 - Virtual Reality Software and Technology*, pages 45–54. World Scientific Publishing Company, 1994.

[SUS95] M. Slater, M. Usoh, and A. Steed. Taking steps: The influence of a walking metaphor on presence in virtual reality. *ACM Transactions on Computer Human Interaction*, 2(3):201–219, 1995.

[Sut65] Ivan E. Sutherland. The ultimate display. In *IFIP Congress Proceedings*, volume 2, pages 506–508, 1965.

[Sut68] I. E. Sutherland. A head-mounted three dimensional display. In *Proc. AFIPS Fall Joint Computer Conference, 33*, pages 757–764, 1968.

[Tri88] L.L. Tripp. A survey of graphical notations for program design: An update. *ACM Sigsoft Software Engineering Notes*, 13(4):39–44, 1988.

[USV96] M. Usoh, M. Slater, and T.I Vassilev. Collaborative geometrical modeling in immersive virtual environments. In M. Göbel, editor, *Proceedings of the 3rd Eurographics Workshop on Virtual Environments, Monte Carlo, 19-20 February*, 1996.

[vRCBF95] F. van Reeth, K. Coninx, S. De Backer, and E. Flerackers. Realizing 3D visual programming environments within a virtual environment. In F. Post and M. Göbel, editors, *EUROGRAPHICS '95*, number 14(3) in Computer Graphics Forum. Blackwell Publishers, 1995.

[VRM96] The virtual reality modeling language specification, version 2.0, iso/iec wd 14772. Available at http://cosmo.sgi.com/moving-worlds/spec/index.html, August 4th 1996.

[WAB93] Colin Ware, Kevin Arthur, and Kellog S. Booth. Fish tank virtual reality. In *Proceedings of INTERCHI '93*, pages 37–42. ACM, 93.

[War90] C. Ware. Using hand position for virtual object placement. *Visual Computer*, 6(5):245–253, 1990.

[Wat93a] R. Watson. A flexible gesture interface. Technical report, Department of Computer Science, Trinity College, Dublin, 18th December 1993.

[Wat93b] R. Watson. A survey or gesture recognition techniques. Technical Report TCD-CS-93-11, Department of Computer Science, Trinity College, Dublin, 1993.

[Wen92] E.M. Wenzel. Localization in virtual acoustic displays. *Presence: Teleoperators and Virtual Environments*, 1(1):80–107, 1992.

[Wex94] Alan Daniel Wexelblat. A feature-based approach to continuous-gesture analysis. Master's thesis, Massachusetts Institute of Technology, May 1994.

[WF94] C. Ware and G. Franck. Viewing a graph in a virtual reality display is three times as good as a 2D diagram. In *Proc. 1994 IEEE Symposium Visual Languages*, pages 182–183, 1994.

[WG89] D. Weimer and S.K. Ganapathy. A synthetic visual environment with hand gesturing and voice input. In *CHI'89 Proceedings*. ACM Press, New York, 1989.

[WJ88] C. Ware and D.R. Jessome. Using the bat: A six-dimensional mouse for object placement. *IEEE Computer Graphics and Applications*, pages 65–70, November 1988.

[WO90] C. Ware and S. Osborne. Exploration and virtual camera control in virtual three dimensional environments. *Computer Graphics*, 24(2):175–183, March 1990.

[WS91] C. Ware and L. Slipp. Using velocity control to navigate 3D graphical environments: A comparison of three interfaces. In *Proceedings of the Human Factors Society 35th Annual Meeting*, 1991.

[Zel92] David Zeltzer. Autonomy, interaction and presence. *Presence: Teleoperators and Virtual Environments*, 1(1):127–132, 1992.

[Zha93] Rui Zhao. Incremental recognition in gesture-based and syntax-directed diagram editors. In *INTERCHI'93 Proceedings*. ACM Press, New York, 1993.

[Zim85] T.G. Zimmerman. Optical flex sensor. US Patent 4,542,291, September 1985.

[ZL87] T.G. Zimmerman and J. Lanier. A hand gesture inteface device. In *Proceedings of SIGCHI/GI*. ACM Press, New York, 1987.

[ZL91] T.G. Zimmerman and J. Lanier. Computer data entry and manipulation apparatus and methods. US Patent 4,988,981, January 1991.