MaPS: Movement and Planning Support for Navigation in an Immersive VRML Browser

John D. M. Edwards and Chris Hand

Department of Computer Science
De Montfort University
The Gateway, Leicester, UK
LE1 9BH

jde@tomedii.demon.co.uk cph@dmu.ac.uk

ABSTRACT

This paper describes the design and implementation of the user interface for a prototype immersive VRML2 browser, with particular reference to the planning and viewpoint movement aspects of navigation in the virtual environment.

Rather than being hard-coded in the browser, the user interface objects are part of the virtual environment itself (*i.e.* stored in the VRML scene graph). Advantages of this "first-class user interface" are described, and implications for an open, extensible approach to user interface evolution and browser implementation are considered.

KEYWORDS

VRML2; immersive browser; first-class user interface; navigation techniques; user interface metaphors.

1. INTRODUCTION

While the VRML specification and the browsers which implement it have both evolved rapidly over the past couple of years, the 2D mouse-based interaction techniques typically provided by browsers are still considered unsatisfactory by many users. Moreover, when we turn to the problem of creating a browser for an immersive VRML system these mouse-based techniques are no longer applicable or desirable. There is therefore a need to investigate user interface design and implementation for VRML browsers, and immersive systems in particular. (In this paper the term "immersive" is used to refer to a system which provides "sensory immersion", *i.e.* uses a head-mounted display with tracking.)

The interaction tasks which a VRML browser's user interface needs to support can be divided into hypermedia tasks (such as following hyperlinks and placing bookmarks) and virtual environment (VE) tasks, such as navigation and object manipulation. This paper concentrates on interaction techniques which support two aspects of navigation: movement and planning.

The structure of the paper is as follows. Section 2 covers some of the requirements and problems of using an immersive system and describes related work. Section 3 then explains what we mean by a "first-class" user interface and describes the advantages of this approach. The theoretical background to our work is outlined in section 4, "Movement and Planning Support", while the implementation of these ideas in a prototype immersive VRML2 browser is covered in some detail in section 5. Section 6 briefly describes some initial evaluations, section 7 discusses future work, and finally section 8 presents some conclusions, including comments on VRML2 and possible improvements to the specification.

2. AN IMMERSIVE INTERFACE

One of the principle aims of this work was to investigate navigation techniques suitable for use with a VRML2 browser running on an immersive platform as opposed to a desktop system. The work described in this paper was carried out using a Virtuality™ Elysium™ Ultrascaler immersive VR system with V-Flexor™ 6 degrees of freedom (DOF) input device and Visette™ head-mounted display (HMD), running in stereo at a resolution of 640 × 480 pixels.

A distinguishing attribute of an immersive system is that it divorces the user's sensory modalities from the real world, which renders a large proportion of conventional input-output devices unusable. Certainly, the wearing of HMDs prevents users from interacting visually with many real world peripherals, such as mice and VDUs. An immersive interface must therefore provide a virtual substitute for these I/O devices if the user is to interact successfully within the virtual world.

2.1 Related Work

Designers of 3D interfaces have recognised for some time that creating a one-to-one mapping between a 6-DOF input device and its representation in the VE has direct benefits in terms of improved spatial understanding during object manipulation, due to the natural kinaesthetic correspondence [17] between hand position and the manipulated object or tool. Projects such as MIT's 3-Draw [12], have demonstrated the efffectiveness of instrumenting familiar tools (such as a clipboard and stylus) with a 6-DOF tracking device, allowing direct input techniques to be used. Navigation techniques have also used this approach: Brook's "shopping cart metaphor" [2], the Delft Scooter [14] and Slater's "virtual treadmill" [13] all provided inherent (and in some cases active) kinaesthetic feedback.

However, while instrumented "props" [5] are useful for specific tasks, there is also a requirement for more general purpose techniques (and, in the case of VRML browsers, for using more

widely-available 3D devices). The solution proposed by Wloka and Greenfield, known as the Virtual Tricorder [19], uses the metaphor of a re-configurable tool and again adopts a direct mapping between input device and its virtual representation. In this case the representation is a model of the actual ultrasonic 3D mouse used, with the addition of extras such as "2D anchored menus" which pop up when required. The Virtual Tricorder was the initial inspiration for the immersive VRML navigation control described below.

Another approach which is related to the work described here is the "world in miniature" (WIM) metaphor [16], an extension of the world-in-hand technique [18] which provides the user with a hand-held miniature version of the virtual world. The WIM technique as described by Stoakley *et al* allowed for both navigation and object manipulation, although controlling the scale of the WIM model was an issue. The WIM technique has also been used in a collaborative, immersive VRML-based system implemented by HITL during phase II of their Greenspace project [9], albeit using a pre-defined model rather than a dynamically updated representation.

3. BUILDING A FIRST-CLASS INTERFACE

3.1 Concepts

Perhaps the most straightforward approach to implementing a browser's user interface would be to code it as part of the browser itself, and this has been the case with most, if not all, VRML browsers to date. However the VRML2 specification, in conjunction with dynamic routing capabilities such as those of the Carmel[1] graphics kernel [15] used in our implementation, makes it possible to instantiate the interface components as "first-class" world objects both in terms of visibility (they could be rendered as part of the normal scene graph) and connectability (they could be connected directly to objects in the environment using ROUTEs). As well as greatly extending the flexibility of the interface and permitting rapid experimentation through ROUTE editing, this approach also contributes to the potential longevity of the system, as it will be largely compatible with other VRML2 products and developments.

Figure 1 highlights the key differences between a conventional browser interface and its first-class equivalent. It should be noted that there is no difference between the scene graph processing abilities of the two browser applications; the contrast between the two is based on the fact that one contains a built-in interface, whereas the other constructs its interface as part of the main scene graph. The first-class interface can make use of a lightweight browser application which comprises little more than a scene graph processor – all of the interface functionality has been migrated into the screen graph itself.

Figure 1(b) shows that the first-class interface still includes a direct connection between the browser application and some "Core" I/O devices. Although it is theoretically possible to build *all* of the interface functionality into scene graph nodes, we don't suggest this as a viable proposition. For example, a VR system's rendering functionality could be built entirely into the behaviour of a Camera node; however, due to hardware specific issues this would be unlikely to yield practical levels of performance. It is therefore acknowledged that it may be necessary to preserve some core I/O functionality as an integral part of the browser application. This core functionality will probably correspond to the baseline requirements of the VRML2 specification.

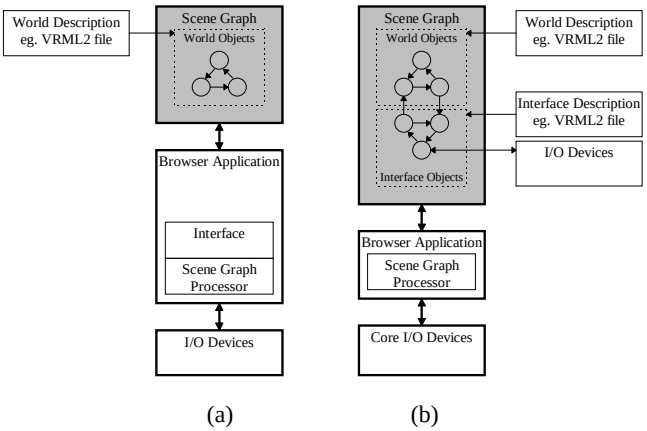

Figure 1: Conventional interface (a) and first-class interface (b)

3.2 Key advantages

The creation of a first-class interface opens up some exciting possibilities in the field of browser interface design. The key advantages of this approach are described below.

Portable

Probably the most significant advantage of constructing the interface within the VRML2 paradigm is that it results in a system which is essentially portable to other platforms.

Reusable and Compact

By developing self-contained nodes which encapsulate key behaviours, a first-class interface is able to make use of one of the key benefits of the object-oriented paradigm: reusability. By hiding the nodes' internal workings and providing a carefully-designed public interface (*i.e.* a node's exposed fields) it is more likely that the node can be successfully connected to other (possibly third party) nodes. This also leads to a more compact browser implementation which is faster to develop and simpler to maintain.

Visible

A user interface (especially an immersive interface) is likely to require some form of visual representation. A first-class interface is able to make use of the visualisation functionality already built into the browser. In the general case there is no reason why a first-class interface should not make use of *any* of the modalities supported by the browser.

There are also situations where aspects of the interface need to be made visible to the rest of the virtual environment, rather than just the user. For example, in multi-user environments, the interface is responsible for generating an avatar which must be visible to other users. Similarly, if the interface supports the notion of alternate viewpoints (*e.g.* map views) it is important that the user's

[1] Developed by Virtuality and Delft University of Technology, Carmel is an extensible object store which supports VRML 2 node types and routes.

position is visible. A first-class interface can take advantage of built-in scene processor functionality to make specific aspects of the interface visible at no extra cost.

Extensible and Connectable
By removing the interface from the browser application, new components may be wired into the interface as and when they are required. VRML2's inherent network portability suggests that interface components could be distributed over the Internet and wired up by users to create custom interfaces. Script node behaviours should make use of a network portable language, such as Java, JavaScript or VRMLScript. Using the VRML2 routing paradigm it is a simple matter to connect both *world* and *interface* objects together in a variety of configurations.

4. MOVEMENT AND PLANNING SUPPORT

This section describes the theoretical background to the navigation techniques designed and implemented in our prototype.

The user's ability to navigate in a virtual space is affected by the virtual environment itself and by the user interface. Environmental aspects (how the space is designed, landmarks and so on) are provided by objects within the virtual world and are generally placed there intentionally by its creator. Any aids to navigation provided directly by the initial environment (before it has been modified by the user) are, by definition, not within the control of the user interface, and therefore outside the scope of this work.

The user interface provides support for the navigational task independently of the virtual environment. Support techniques may be divided into two main sub-groups: Viewpoint Manipulation Tools which contribute to the tactical component of navigation (*Movement*), and Navigational Aids, such as maps and compasses, which assist in the more strategic aspects of wayfinding (*Planning*). A navigational user interface should provide support for both aspects of the navigational task, hence the acronym MaPS (Movement and Planning support).

Sections 4.1 and 4.2 examine these two aspects of navigation while section 5 describes the practical implementation of these ideas.

4.1 Movement

A fundamental requirement for navigation in a virtual environment is a facility to manipulate the effective viewpoint, that is, to move. Gale *et al* [4] note that the acquisition of spatial knowledge, essential for wayfinding, is primarily based on "direct environmental experience" which is usually gained via movement.

In considering the many metaphors for viewpoint manipulation, it is possible to make a further classification based on the effective frame of reference. Some techniques are analogous to moving a viewpoint through a world (*egocentric*), while others allow the observer to manipulate the world in front of a static viewpoint (*exocentric*). In general, navigation through an environment is associated with an egocentric viewpoint. Examples of egocentric movement include the "eyeball-in-hand" and "flying vehicle" metaphors [18].

Movement metaphors may also be classified depending on what we might call "natural" movement as opposed to "vehicle" movement. Natural movement is often characterised by a one-to-one mapping between movement of the input device and its manifestation in the virtual environment. A tracked head-mounted display unit represents possibly the most natural of all viewpoint manipulation techniques, as changes in viewpoint correspond directly to movement of the user's head. Similarly, world-in-hand and eyeball-in-hand viewpoint manipulation techniques also rely on one-to-one mappings between the position of the input device position and the viewpoint. Vehicles, on the other hand, tend to utilise more complex mappings between the input device and the resultant movement in the virtual environment.

4.2 Planning

In addition to providing the user with the ability to move through an environment, essentially a tactical exercise, the interface is also able to provide the user with a set of tools to assist in the strategic aspects of wayfinding, that is, planning. Trailblazing and the use of maps are both examples of planning tools for navigation.

4.2.1 Trailblazing

Trailblazing involves leaving physical markers in the environment as a way of marking or encoding locations. This may add meaning to the environment, or just make places easier to find in the future. The markers are similar to simple landmarks, with the important difference that true landmarks are created by the designer of the environment, whereas trail marks are created by users.

Darken and Sibert [3] describe the use of elevated (for visibility) coloured cubes as visual markers, or "virtual breadcrumbs", which may be dropped by the user as a means of assisting navigation. The mechanism was provided for trail making and a facility existed for dropping breadcrumbs automatically at a preset frequency. However, an informal subject evaluation revealed that users dropped breadcrumbs manually as a way to label the environment with their own landmarks. Depending on how the breadcrumbs were dropped they could provide a variety of useful information, such as marking an area as visited, delineation, and recording a change in direction

The idea of trailblazing may also be extended to include the notion of "embodied bookmarks". Just as a hypertext system allows a user to record (and return to) a set of HTML document positions, a VRML browser should allow the user to record and return to a set of viewpoints. The process is analogous to the creation of a set of custom hyperlinks, in which case it would appear logical to embody these bookmarks, just like conventional hyperlinks. This could be achieved using a trailblazing technique.

4.2.2 Maps

Spatial cognition research identifies two distinct kinds of spatial knowledge: *Route Knowledge* enables navigation from point to point using landmarks, and is based on an egocentric frame of reference. *Survey Knowledge* (or map knowledge) on the other hand enables efficient planning of journeys not previously encountered, and is based on an exocentric reference frame.

Survey knowledge has been shown to be essential for effective wayfinding [7], and although it can be derived from extensive route knowledge, it is available directly from a map. In most situations, the ability to rapidly obtain survey knowledge makes maps an invaluable tool for navigation in a real environment and this is also the case in a virtual environment; as might be expected, the subjects whom Darken and Sibert provided with a map showed significant improvements in navigational ability over those without [3].

The design of maps for virtual environments can draw on real world map design principles [6]:

1. *Two-point theorem*: it must be possible for the user to relate two points on the map with two points in the environment.
2. *Alignment principle*: the map should be aligned with the terrain.
3. *Forward-up principle*: the upward direction on a map should align with what is in front of the user.

Implicit in these design principles is the requirement that it should be easy for the user to relate their current view with a position and orientation on the map. The principles are actually easier to achieve with a virtual environment, as the map alignment and an indication of current position can be generated dynamically. The alignment principle can be modified to suit specific navigational tasks. Empirical evaluations [3] demonstrate that a view-aligned map (the map turned with the view) is more effective for exploration, whereas a terrain-aligned view (the map remained stationary) maintains a more consistent cognitive map of the overall environment. There is no reason why this should not be switchable depending on the search task.

The next section describes the development of a number of new VRML2 nodes which may be combined to produce a variety of map views.

5. IMPLEMENTATION

This section describes the implementation of movement and planning support in the context of an immersive VRML2 browser.

Movement of the viewpoint was supported by a Navigator node, described in section 5.2. Planning was assisted by a map-type tool and a trailblazing technique. Both movement and planning were achieved through a programmable tool similar to the Virtual Tricorder, which may be easily re-configured to provide many different navigational aids. Since the main area of focus in our investigations was this tool and its map modes, the implementation of these is described in detail in sections 5.1 to 5.7.

A simple trailblazing technique was implemented by using a simple menu (implemented as an in-built node type) to instruct the scene graph manager to instantiate a new object just behind the user's current position. While this was relatively simple to implement, a more complete solution would allow the user to choose between a variety of different objects, thus allowing the user to add distinct custom landmarks to an environment. Lack of space precludes any further discussion of the trailblazing tool or the Menu node.

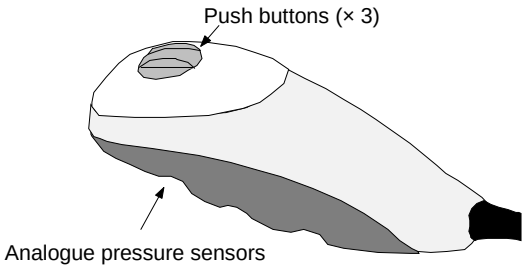

Figure 2: V-Flexor 6 DOF Input Device

5.1 VirtualFlexor: An Embodied Input Device

The Elysium system's V-Flexor input device (Figure 2) is ideally suited to the implementation of a "virtual tool" style of control, through the creation of the VirtualFlexor. The VirtualFlexor is effectively a virtual model of the real V-flexor, with a flattened cuboid attached to the top of its handle. As the real device is moved, so the VirtualFlexor follows. The attachment may be used as a virtual screen which can act as a focal point within the immersive interface, for example displaying map views or menu choices. The VirtualFlexor also serves as a movement tool via the Navigator node.

A script node was responsible for reading the position and orientation of the physical device. The translation and orientation fields of this script node were then routed directly into a VRML2 transform node which contained a shape node defining the geometry of the VirtualFlexor (Figure 3).

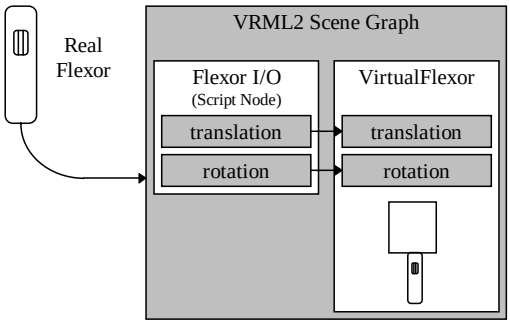

Figure 3: The VirtualFlexor

(The Flexor-I/O node could be re-used for a variety of tasks. For example, the interface could adopt the "world in hand" metaphor simply by sending two ROUTE statements to the kernel.)

Additional input parameters are available from the *real* V-Flexor via a four-way analogue pressure sensor placed in the hand grip and three two-state push buttons situated on top of the device. These inputs are also made available to the scene graph by the Flexor I/O node.

5.2 Navigator node

The principle task of the Navigator node is to allow users to manipulate their effective viewpoint. The outputs of the Navigator were connected to a flyable platform or "magic carpet"; by ensuring that the Camera (the user's viewpoint) and the

VirtualFlexor were members of the platform's transform, the user was given the impression of travelling on a virtual vehicle. The user was therefore able to move their head and hand independently of the direction of travel. This configuration takes advantage of the natural constraints afforded by the "cyberspace metaphor" [18], particularly its ability to orient the user in relation to the vertical axis. Murta maintains [10] that users' perceptions of the vertical axis are critical to their ability to make sense of a virtual environment.

The Navigator node is essentially a tool for generating geometric transformations and aims to centralise all of the movement functionality required by the interface on a single node. The Navigator is able to encapsulate a number of different mappings between the input device and the resulting transform dynamics. Transforms generated by the Navigator are presented as translation and rotation output fields, which may be routed into any suitable node; this routing arrangement was chosen to enable the translation and rotation components of a transformation to be isolated. Such isolation would not be possible if objects were required to accumulate transformations from the Navigator via composition.

The simpler types of navigation, for example walking and flying, represent a relatively straightforward re-mapping of the inputs from the V-Flexor and HMD. Point-of-interest navigation [8] and hyperlinking involve rather more complex algorithms. Techniques for enacting the following of hyperlinks have been investigated but not yet implemented (this functionality would probably be incorporated as part of the Navigator node). Although this is an important issue for VRML, further discussion of hyperlinks is outside the scope of this paper.

5.3 TextureCamera Node

The TextureCamera node is based on the standard VRML2 Camera[2] node. Whereas the Camera node produces an image on the main display, be it HMD or VDU, the TextureCamera node renders its view to a texture (*i.e.* bitmap) within the environment. Like the Camera node, the TextureCamera provides parameters for controlling its transformation and field of view (zoom).

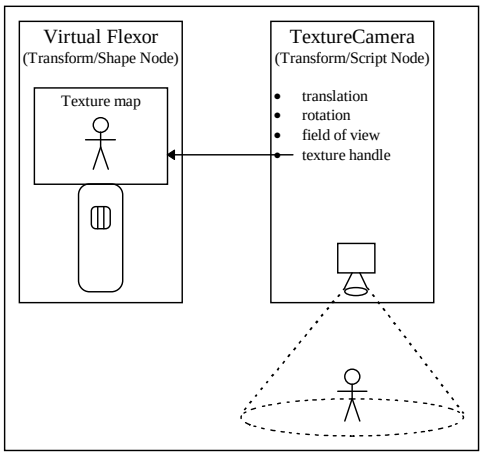

Figure 4: TextureCamera node

[2] As of the final VRML 2.0 specification, this is now known as Viewpoint.

The destination texture could, of course, exist on any object (or objects) within the virtual environment. However, for the purposes of the navigational interface the TextureCamera's view is rendered onto a screen attached to the VirtualFlexor, as shown in Figure 4 and Figure 5.

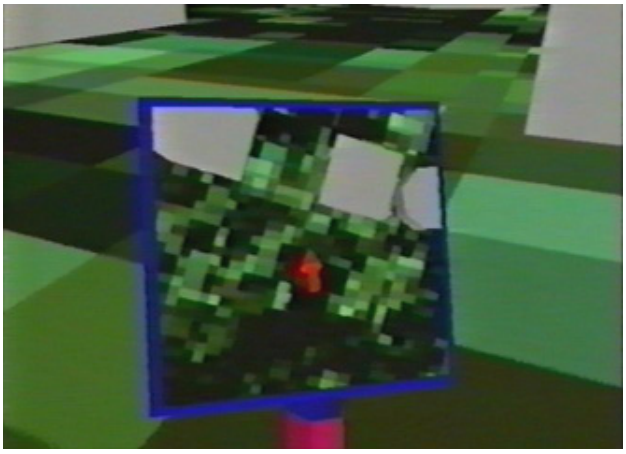

Figure 5: VirtualFlexor Displaying Map View

The implementation of the TextureCamera makes use of an advanced facility of Virtuality's 3D graphics hardware and software, and is effectively coded as a built-in node. It would be difficult, if not impossible, to produce the same effect using standard VRML2 script nodes. However, the process of adding the same functionality directly to a VRML2 browser would be much simpler, as it would probably only require that the render destination of the scene be set to a texture map, rather than the main view. Given that the TextureCamera opens up such a wide range of possibilities in terms of providing alternate views within a virtual environment, it would seem reasonable to propose such a node for consideration in future revisions of the VRML specification.

5.4 Pointer node

The Pointer node was developed to provide an automatic orientation mechanism for the TextureCamera node. When supplied with the absolute translations of source (*e.g.* the camera) and target (*e.g.* the user), the Pointer node is able to point the target at the source. More precisely, the Pointer node generates a rotation which aligns the line-of-sight of the source with an imaginary line running between the co-ordinate origins of the source and target.

This arrangement leaves one degree of freedom unaccounted for, namely the rotation of the source around the line-of-sight to the target. This is accommodated by the addition of a viewup_rotation field which dictates the orientation of the top of the view. For example, if the pointer was controlling a camera positioned above the target, then connecting the target rotation to the viewup rotation would result in a "view-oriented" map view.

Figure 6 shows how the Pointer could be used to orient a TextureCamera towards a target in the virtual environment, in this case the user.

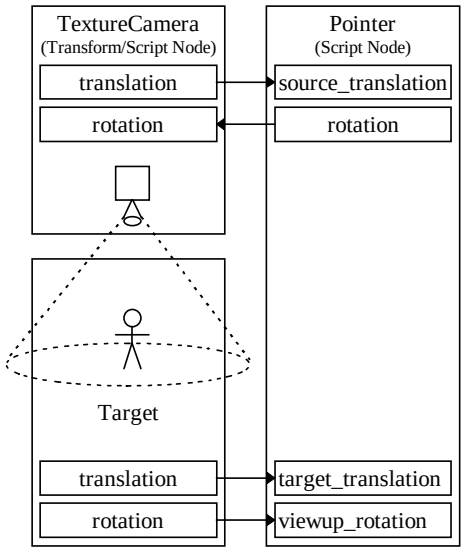

Figure 6: Pointer node

The process of designing this node has suggested a possible enhancement to the VRML2 specification. In order to successfully calculate the necessary orientation, the Pointer node requires the *absolute* translations of source and target. However, the VRML2 specification is based on cumulative transformations and only the *relative* translation of a given Transform node is directly available. Provided that the participating nodes are located at the root level of the scene graph, this presents no problem, but this is not likely to be always the case.

For example, it would be quite reasonable to want to point a camera attached to a building at a man on a horse. Unfortunately, the camera's translation is (naturally) relative to the building and the man's is (naturally) relative to the horse; in short, the Pointer will not know which way to turn.

It may be possible to calculate absolute transforms from within a Script node (for example, using VRMLScript), however this is likely to be quite involved and computationally expensive. Since any scene graph processor must determine all accumulated transformations every time the view is rendered, it should present little overhead to make this information available as EventOut fields in each Transform node.

For our implementation, the behavioural aspects of the Pointer node were implemented as a C++ class within the Carmel kernel. It would be relatively straightforward to implement this functionality using a portable language such as VRMLScript from within a VRML2 script node, were it not for the aforementioned problems with accumulated transforms. An interim solution would be to ensure that participating nodes are all members of the root transform.

It is worth noting some other potential applications of the Pointer node. For example, it could be used in conjunction with the Navigator node to orient the user towards another object in the environment, perhaps a hyperlink. Alternatively, a collection of Pointer nodes could enable the audience of a virtual tennis match to keep their eyes on the ball.

5.5 Slider node

Another requirement was that the user be given control over the field of view (zoom) of the camera. However, there was a fundamental mismatch between the data available from the input device (two-state buttons) and the analogue value required to control the zoom.

The Slider node was developed to bridge this gap and its operation is conceptually similar to a sliding potentiometer, such as a volume control on a radio.

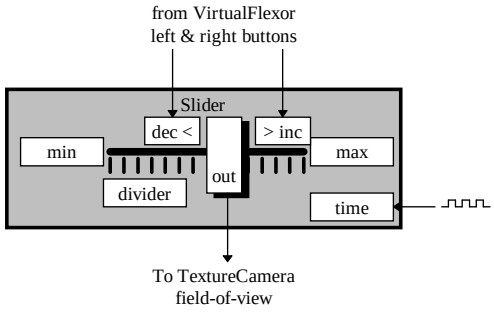

Figure 7: Example use of Slider node

The Slider provides real number fields to specify the range of output values (min, max). It also allows for the number of divisions within this range to be set as an integer (divider).

The Slider is controlled using three further fields. Slider operation is synchronised by providing a timing pulse (time). Every time a "clock pulse" is received, the sliders output (out) is modified according to values at the Boolean directional inputs (inc, dec). The Slider is, of course, reusable in any situation requiring potentiometer-style control.

5.6 User representation

The two-point theorem [1] states that the map reader should be able to match two points on the map with two points in the environment. This task is simplified if the user is given some indication of their position in the environment on the map view. When generating the map view by means of a camera view it is necessary to generate a user embodiment or avatar. A large red arrow was chosen as a short term solution, as this could indicate both the user's position and orientation and would be visible from a distance. One possible enhancement would be to link the arrow size to the TextureCamera's field of view, ensuring that the arrow always appears at an optimal size in the map view.

For certain camera configurations it may be preferable to use a more sophisticated avatar which closely reflects the user's physical attributes, possibly using a jointed model of a human body. However, from large camera ranges (*e.g.* Map views) such an embodiment would be too small to be seen.

5.7 A Navigation Metaphor Construction Kit

By combining the VirtualFlexor, Navigator, TextureCamera and Pointer nodes in various ways it is possible to generate a variety of useful configurations which comprise many different navigational metaphors, described briefly below. Any one of these

configurations can be set up quickly and easily by configuring the appropriate routes, for example via menus on the VirtualFlexor's screen. Another option would be to change the screen from a flattened square to (say) a cube, providing several surfaces which could be used to present a subset of these techniques.

Map view (North-aligned). The top edge of the map view will always be aligned with virtual "North", so that when the user turns around, the map will stay still and the user's position indicator (that is, the avatar) will rotate. The scale and perspective of the map view may be altered by adjusting the height of the TextureCamera above the user and its field of view (possibly via a Slider node).

Map view (View-aligned). This map view utilises the forward-up equivalence principle [1] to ensure that the upward direction on the map shows what is in front of the user.

(The North-aligned and View-aligned map views were the main focus of our initial evaluations.)

Rear-mounted Camera. In this example, the TextureCamera is positioned above and behind the user. The view is aligned with the positive vertical axis, as one would normally expect from a camera view. This choice of view alignment is likely to contribute to vertical axis awareness [10] and should help the user to make sense of the view.

Targeting Satellite Camera. This camera may be moved relative to the user's position using the functionality provided by the Navigator node. Wherever it is, it will always point at the user, with an up-aligned view. It is really just an extension of the Rear-mounted Camera, which affords the user greater control over where they position the camera. Obviously, while the Navigator is connected to the TextureCamera, the user will have no way of moving themselves.

Autonomous Satellite Camera. A variation on the previous Targeting Satellite Camera, this camera dispenses with automatic tracking, instead allowing the user to control its orientation using Navigator node functionality.

Targeting Drone Camera. This configuration is the same as the Targeting Satellite Camera but releases the TextureCamera so that it is free to move independently of the user. Among other things, this allows it to make sensible use of the hyperlink capabilities of the Navigator node. For example, in the case of *intra*-world links, it may be useful to be able to send a drone down a hyperlink to stare back at you from the other end. This may contribute to the user's ability to form a cognitive map of the spatial relationships between two hyperlinks connecting points within the same world.

Autonomous Drone Camera. A variation on the Targeting Drone Camera, this camera dispenses with automatic tracking entirely, allowing the user to control its orientation using Navigator functionality. Unlike the Targeting Drone Camera, this configuration can also be applied to *inter*-world hyperlinking, allowing the user to make a useful reconnaissance of a remote (and spatially unconnected) virtual environment.

Eyeball. A variation on the eyeball-in-hand metaphor [2][18], the camera orientation is linked directly to the orientation of the VirtualFlexor. In this example, the camera is situated high above the user. The resulting view would probably be more useful if the Camera view was diverted to a HUD (head up display, *i.e.* superimposed on the graphics display in the HMD), rather than the VirtualFlexor screen, as the user is likely to experience difficulties in synchronising view position and screen position.

Giant. A similar idea to Eyeball, except that the orientation of the TextureCamera is directly related to the orientation of the user's HMD. What this provides, therefore, is the view that the user would have if they were the same height as the TextureCamera. This has many similarities to viewpoint manipulation techniques which allow the user to increase their size in order to get a map-like view of an environment. This configuration has the advantage that the user doesn't need to change size (which could be disconcerting for other users in the environment). Moreover, both views are simultaneously visible. It may be beneficial to introduce an additional transformation into the User-TextureCamera hierarchy, so that the Giant view can be offset and oriented slightly towards the ground. The provision of a HUD option would also be useful.

Rear-view Mirror. This configuration effectively attaches the TextureCamera directly to the user Camera with a rotational offset of 180° around its vertical axis. The resulting view therefore displays the scene exactly opposite to the user's main view, giving them the option of "having eyes in the back of the head".

Hand-Held Mirror. By using a similar technique to the Rear-view Mirror, it is possible to turn the VirtualFlexor into a hand-held mirror. Among other things, the user's own avatar may be examined in this mirror. The capacity for self-examination may assist in the processing of establishing the user's presence in the environment, or may be useful in the growing area of research into the visualisation of body image in VEs (*e.g.* [11]).

Magnifying Glass. Almost identical to the hand-held Mirror, but without the rotational offset, this virtual device mirrors the operation of a real world magnifying glass, except that by giving the user control of the field of view (via the Slider) the device also becomes zoomable. This technique replicates the functionality of the Virtual Tricorder's "magic lens" [19].

6. INITIAL EVALUATION

A small number of initial user trials were run to compare the North-aligned and View-aligned map tools. The scenario used for testing was a simple maze-running task using a small maze comprised of cubes, cones and cylinders each of which could appear at two different heights. Subjects were placed outside a randomly-generated maze and instructed to first find the centre, then to find three adjacent short cones, and finally to reach the entrance/exit once more. Subjects were instructed to perform these tasks as fast as possible but without crashing into maze objects. Five subjects were tested with three conditions: no map, North-aligned map and View-aligned map.

When queried at the end of three trials, all subjects claimed to prefer the view-aligned map tool as a means of navigation. Although some subjects found it a useful strategy to plan a route using the map, and then move it out of the field of view while moving, others relied very heavily on the map, to the extent that they neglected to look where they were going, increasing collisions with the maze. (The cost of concentrating on a map to the exclusion of the external environment is clearly much less in a VE than it is, for example, walking down a busy city street.)

Finding the three short cones in the maze required the interpretation of perspective from the map view, and this was easily achieved by moving the VirtualFlexor closer to the face to increase available detail (Figure 8).

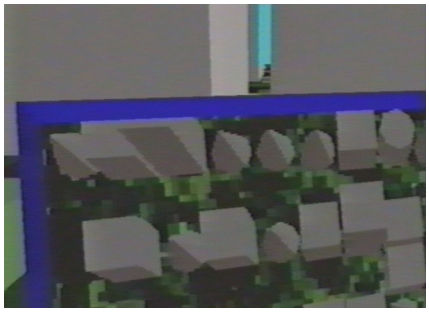

Figure 8: Close-up View of Map Showing Perspective

One subject (the only one who was a novice VE user) claimed that the map view tended to "get in the way", apparently finding it difficult to manipulate the V-flexor's buttons in any orientation other than directly to the front, and was therefore reluctant to move the map out of the field of view while moving. This subject suggested that the map could be fixed to the main view, in which case it becomes a head-up display. This "degenerate" case could also be used for non-immersive systems.

7. FUTURE WORK

Future work will involve further investigation of the re-configurable navigation techniques described in section 5.7, along with ways of changing the movement metaphor. These are technically simple, only requiring changes to a few routes, but the user interface side is less straightforward. For example, our initial experiments with text-based menus proved problematic, due to the commonly-experienced difficulty of reading text in a HMD. A single tool overloaded with many functions is likely to become difficult to control, so it may be fruitful to use alternate modalities such as speech in this control task.

From an architectural point of view, the VRML community's discussions of issues such as external software interfaces have been progressing in parallel with this project, and the outcome of these will be taken into consideration in planning future work.

The notion of alternate viewpoints, particularly those outlined in the description of the Satellite and Drone varieties of Camera, presents the possibility of "recursive immersion". It would be interesting to examine the effects of these scenarios on the user's feeling of presence. For example, where would a user flying a Drone Camera perceive their presence to be?

8. CONCLUSIONS

The VirtualFlexor described in this paper may be considered a descendent of Wloka's Virtual Tricorder and the WIM technique of Stoakley *et al*. However, unlike the WIM model, the VirtualFlexor may be easily scaled, or reconfigured into a different tool entirely. Unlike the Tricorder, the VirtualFlexor provides no object manipulation functions, but provides a wider range of navigation support tools.

Our experiences with VRML2 during implementation uncovered some ways in which the specification could be improved or augmented:

- The advantages of providing absolute positional information from Transform nodes (section 5.4)
- New node type: TextureCamera (section 5.3).
- In order to allow sensible progressive parsing, any fields which affect a node's children should be specified *before* those children.

The VirtualFlexor is a good example of the benefits of developing first-class interface objects, as the application of this methodology resulted in a simplified implementation task. To have implemented the VirtualFlexor node as a part of the browser application would have required hard-coding its appearance and behaviour into the browser code, resulting in a less flexible solution.

Finally, the use of portable first-class user interface objects would give all VRML users the option to become creators of user interfaces. We would hope that making the user interface open and extensible would accelerate the evolution of user interfaces for VRML browsers (and 3D interaction techniques in general), provide a natural platform for prototyping and experimentation, and contribute further to the sharing of results and experience within the VRML community.

9. ACKNOWLEDGEMENTS

The authors would like to thank Paul Beskeen, Paul Cameron, Howell Istance, Richard Jacklin, Arnold Paalder, John Rowland, Pieter Stappers, Hugh Steele and Anna R. Thomas for their help and support during this project.

The prototype made use of vrml2parser.zip, supplied to the VRML community by Silicon Graphics, Inc.

This work was supported by Virtuality Ltd. "Virtuality", "Elysium", "V-Flexor" and "Visette" are trademarks of Virtuality Group plc.

REFERENCES

[1] Boff, K.R. and Lincoln, J.E. *Engineering Data Compendium: Human Perception and Performance*. Wright-Patterson AFB, Ohio, USA, 1988.

[2] Brooks, F. P. "Grasping Reality Through Illusion – Interactive Graphics Serving Science". *Proceedings of CHI'88*, May 1988. pp1-11.

[3] Darken, R. and Sibert, J. “A Toolset for Navigation in Virtual Environments”. *ACM User Interface Software and Technology*, 1993, pp157-165.

[4] Gale, N., Golledge, R., Pellegrino, J.W. and Doherty, S. “The Acquisition and Integration of Route Knowledge in an Unfamiliar Neighbourhood”, *Journal of Environmental Psychology*, 10, (1990), pp3-25.

[5] Hinckley, K., Pausch, R., Goble, J. C. and Kassell, N. F. “Passive Real-World Interface Props for Neurosurgical Visualization”. *Proceedings of CHI’94*, April 1994. pp452-458.

[6] Levine, M., Jankovic, I. N. and Palij, M. “Principles of Spatial Problem Solving”. *Journal of Experimental Psychology: General*, 111(2): 157-175. (1982)

[7] Lynch, K. *The Image of the City*. Cambridge: MIT Press, 1960.

[8] Mackinlay, J. D., Card, S. K., and Robertson, G. “Rapid Controlled Movement through a Virtual 3D Workspace”, *Computer Graphics* 24(4), August 1990.

[9] Mandeville, J., Davidson, J., Campbell, D., Dahl, A., Schwartz, P. and Furness, T. “A Shared Virtual Environment for Architectural Design Review”. *Proceedings of Collaborative Virtual Environments ’96*, University of Nottingham, UK, 19-20 September 1996.

[10] Murta, A. “Vertical Axis Awareness in 3D Environments”. *Proceedings of the Framework for Immersive Virtual Environments ’95*, London, UK, 18-19 December 1995, pp.169-176

[11] Riva, G., Bolzoni, M. and Melis, L. “Effects of Immersive Virtual Reality on Body Representations”. *Proceedings of the 3rd UK VR-SIG Conference*, De Montfort University, Leicester, 3rd July 1996. pp121-132.

[12] Sachs, E., Roberts, A. and Stoops, D. “3-Draw: A Tool for Designing 3-D Shapes”. *IEEE Computer Graphics and Applications*, November 1991, pp18-26.

[13] Slater, M., Steed, A. and Usoh, M. “The Virtual Treadmill: A Naturalistic Metaphor for Navigation in Immersive Virtual Environments”. *Proceedings of the Eurographics Workshop on Virtual Reality*, Barcelona, Sept 1993. pp71-83.

[14] Smets, G. J. F., Stappers, P. J., Overbeeke, K. J. and van der Mast, C. “Designing in Virtual Reality: Perception-Action Coupling and Affordances”. In K. Carr and R. England (Eds), *Simulated and Virtual Realities: Elements of Perception*, Taylor & Francis, 1995. pp189-208.

[15] Steele, H. “Re: Dynamic Routes in VRML 2” Message posted to VRML mailing list, 6th August 1996. <URL: http://vag.vrml.org/www-vrml/archives/vrml.9608.gz>

[16] Stoakley, R., Conway, M.J., and Pausch, R. “Virtual Reality on a WIM: Interactive worlds in miniature”. *Proceedings of CHI ’95*, March 1995.

[17] Ware, C. “Using Hand Position for Virtual Object Placement”. *The Visual Computer* 6:245-253, 1990.

[18] Ware, C. and Osborne, S. (1990). “Exploration and Virtual Camera Control in Virtual Three Dimensional Environments”. *Proceedings of the 1990 Symposium on Interactive 3D Graphics* (Snowbird, Utah, March 1990). In *Computer Graphics* 24(2): 175-183.

[19] Wloka, M.M. and Greenfield, E. “The Virtual Tricorder”. Technical Report CS-95-05, Department of Computer Science, Brown University, Providence RI, USA, March 1995.