A Dataflow Representation for Defining Behaviours within Virtual Environments

Anthony Steed
Queen Mary and Westfield College
Department of Computer Science
London
steed@dcs.qmw.ac.uk

Mel Slater
University College
Department of Computer Science
London
M.Slater@cs.ucl.ac.uk

Abstract

Construction of immersive virtual environments usually takes place outside the virtual environment in configuration files or application code. The system presented in this paper allows interaction with and behaviours of objects to be defined whilst immersed within the system by manipulating a dataflow representation of the dialogue occuring between the input devices and virtual objects. A concrete example is presented that illustrates the flexibility and customization opportunities that this approach provides.

1. Introduction

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 description 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) 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 participant with the environment are usually hard-coded within the application that generates the virtual environment. Various toolkits have been designed to assist with 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.

However, the usual method for programming these 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.

The main idea behind this work is to create a virtual environment within which it is possible to both experience an environment and make changes to its underlying interactions and behaviours without having to leave it.

The next Section describes the motivation behind creating such a system. Section 3 describes related work in virtual reality and visual programming. Section 4 explains the underlying model to describe environments. Section 5 describes a simple example table tennis application and Section 6 presents the conclusions.

2. Motivation

Previous work [9, 8] has shown that the best metaphors to employ for interaction in an 3D environment depend on the task being undertaken and the person performing the task. For example compare three navigation metaphors implemented for an immersive system:

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 in place” (virtual treadmill metaphor.)

Flying in the direction of hand pointing is a useful metaphor it since allows the participant to fly around an object whilst their gaze remains fixed upon it. Unfortunately this can be quite tiring for the participant’s arm 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.

However the hand is still involved in the action of walking since the participant has to press a button so the third metaphor relieves this by using a whole body gesture of walking in place.

Not only do each of these metaphors fit particular tasks well, but their usage can affect the level of presence a person might have in an environment. For example, flying in the direction the hand is pointing might be best when exploring an abstract data set, but for a building evacuation training application the virtual treadmill might be better since walking in place leads to fatigue which might be a necessary component of the training.

Thus in the construction of a new environment we have a set of possible interaction metaphors to use and making these manipulable from within the environment means that we can rapidly prototype new environments as well as customize existing ones.

In the system described in this paper we have built a immersive environment within which it is possible to do this, by manipulating an immersive representation of the dataflow between the sensor devices and the tools and object behaviours that exist within the environment.

3. Related Work

Essentially the system is a hybrid visual programming language and object hierarchy. The dataflow, 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.

In describing current visual programming systems for virtual reality we have to distinguish between:

- Visual languages to *describe* virtual environments.
- Visual languages *within* virtual environments.

The first type of system extends the usage of data flow languages for user interface description to describing the interface of the virtual reality system. Body Electric from VPL is such a language [4, p 212-219]. Body Electric ran on a Machintosh adjacent to the main virtual environment generator. Data flowed 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 [7].

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

Producing 3D animations is the aim of at least two visual programming systems which illustrate different directions these tools can take. Virtuality Builder II [3] is an integrated 3D environment in which constraint and object paths can be specified to create animations. VPLA [5] 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, modeling operations, deformations, particle systems and recursive procedures.

Two virtual environment languages for programming that have been implemented are CUBE [6] and Lingua Graphica [10]. CUBE allows visualization of expressions in a functional language, and some limited editing through a desktop based interface. Lingua Graphica is a 3D editor for a C++ based 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. Once a procedure has been constructed it can be written out directly in a text form that is compilable.

The union of these two types of environments would be a visual language within a virtual environment to describe a virtual environment. dVISE from Division [1] is an example of a system that goes some way towards such a system in that there is an immersive menu driven interface that allows editing of object properties and simple behaviours.

4. Environment model

The basic dataflow model consists of three elements: firstly data sources that produce streams of data based on external events, secondly filters that accept one or more input data streams and generate one or more output streams, and thirdly receptors that accept input streams and perform some action in the environment modelled by the dataflow.

Examples of these element for a immersive virtual environment are:

1. Sources: position trackers, joystick buttons and glove data.
2. Filters: gesture recognition, constraints and collision detection.
3. Receptors: virtual tools such as pick and fly and object behaviours such as lights turning on.

Figure 1 shows the abstract definition of the virtual treadmill metaphor implemented with such elements.

The four components of the diagram are:

- The head tracker data source that returns position information.
- The participant object is the representation of the participant within the environment and includes specification of the viewpoint.
- The gesture filter which takes two inputs, a position stream from which to recognize a gesture and a gesture specification and reports success on a binary output stream.

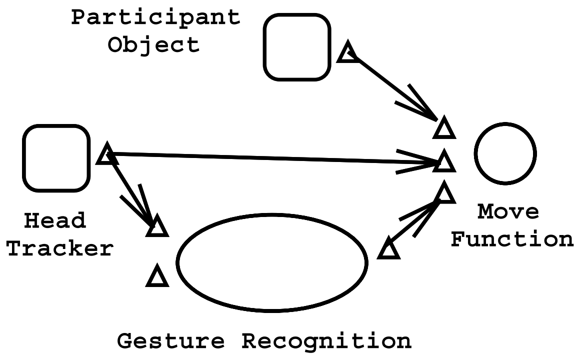

Figure 1. The virtual treadmill metaphor

- The move receptor that takes an object identifier, a position and an enabling binary stream.

In effect Figure 1 means: move the participant in the direction that they are looking when they are making the walking in place gesture.

4.1. Immersive Representation

Inside the environment the dataflow is modelled by a node and arc diagram with each source, filter and receptor being a separate object in the environment having a set of sub-objects representing the input and output connection points. Input and output connectors are joined by stretchable tubes which can be dragged and snapped to appropriate connectors of the same type. The immersive presentation of the virtual treadmill, similar to its abstract description, is shown in Figure 2.

The basic environment is built from a collection of such dataflow segments, defining global functions such as navigation, picking, selecting, object copy and object delete.

4.2. Hierarchy

The amount of detail present in the basic environment description is already quite large, so additional tools and methods are used to hide currently unimportant information. The basic approach is to create an meta-object that encapsulates a collection of objects. This new meta-object can have an arbitrary subset of the input and output connections of its sub-objects as its own connections and in particular it has a special output node that returns an identifying handle so that it can be connected and acted on by nodes in the dataflow model (for example the participant object in Figures 1 and 2).

Inside the virtual environment a meta-object can be displayed at several levels of detail, effectively hiding information unnecessary at the time. Several levels of detail in the definition of a button are shown in Figures 3 and 4. At the

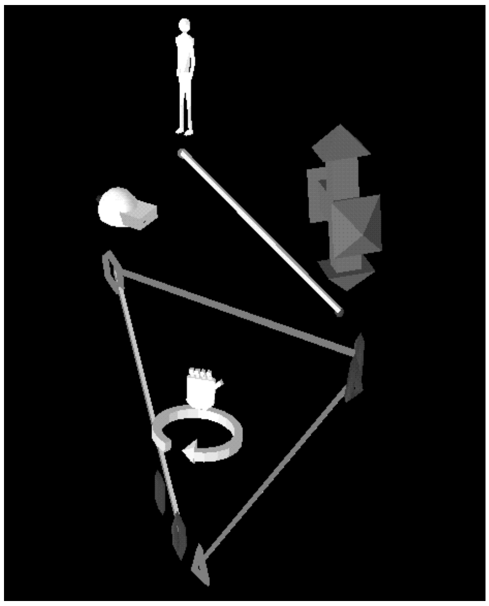

Figure 2. Immersive representation of the virtual treadmill metaphor

lowest level of detail the button is just an object within the environment. The next level up shows that it generates an output that is connected to another object. And at the highest level we see that it generates an output when an object collides with the geometry of the meta-object. The dotted ellipse in Figure 3 indicates the detail normally hidden.

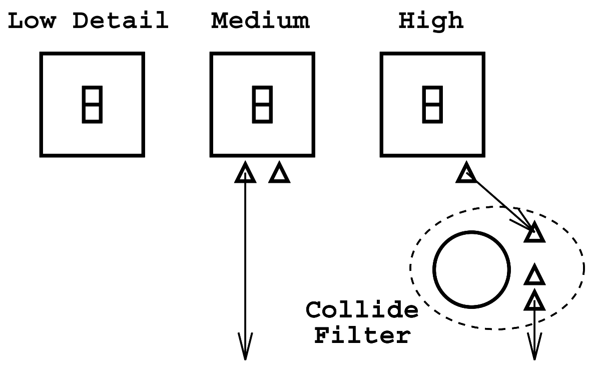

Figure 3. Abstract levels of detail of the button objects

5. Example Table Tennis Application

A virtual environment application is thus described in a dataflow representation that is an integral part of that ap-

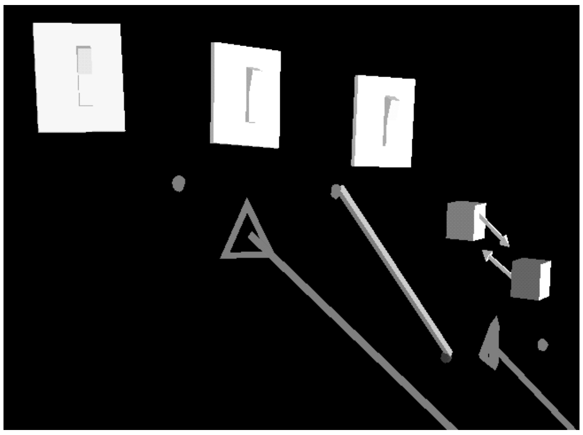

Figure 4. Immersive presentation of levels of detail of the button Objects

plication. In reality on entering the application all the detail of the dataflow would be hidden inside encapsulating meta-objects, which could be examined when modifications needed to be made.

This Section illustrates the components and possibilities for modification of an example table tennis application.

The current system was implemented on a Division Provision 100VPX running dVS version 2.0.

5.1. Components

The major components of the table tennis application, shown in Figure 5, are:

- The participant who encompasses 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 which reset the ball when the ball collides with them.
- The opponent which is 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 dataflow objects. The ball meta-object is illustrated in Figures 6 and 7 with its major components, the dynamic behaviour receptor and a collide filter. The basic version of the dynamic behaviour receptor takes five inputs, identity of object to control, notification of the identities of colliding objects, a reset flag, time values and a gravity strength value. The ball meta-object is connected

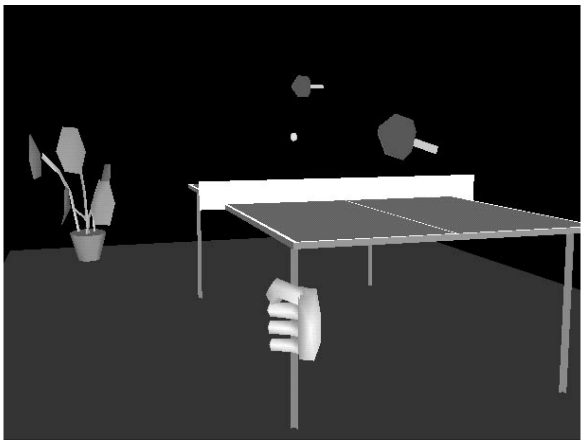

Figure 5. The table tennis application

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.

Everything within the dotted ellipse in Figure 6 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. In Figure 7 the 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.

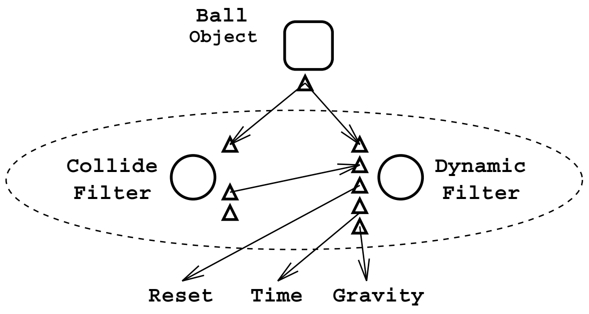

Figure 6. Abstract definition of the ball object

The reset input both initializes the dynamics behaviour and stops it and returns the object to its original position. This corresponds to serving the ball and the reset input can be connected to many sources.

Of the other components of the environment, the net and floor are in effect buttons that are also connected to the ball's dynamic behaviour in order to reset it.

5.2. Customization

Possibilities for customization are many and various. Any of the objects can be changed and copied so there could

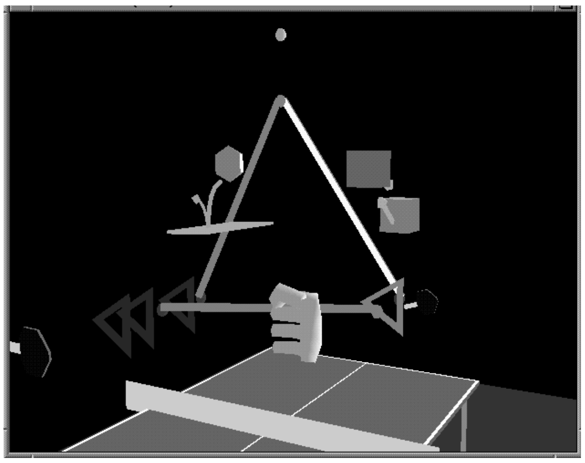

Figure 7. Immersive definition of the ball object

be two balls shaped like plants for example. Thus table tennis is a limiting description of the application since the components could be re-arranged and modified to play volleyball for example, though editing of the geometry of the objects themselves is not yet possible in this system.

More interesting customizations arise when considering how the ball is served, i.e. 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 recognizer. 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 serve position.

Another customizable component of the application is the opponent: an application specific behaviour that knows about the ball and table (and possibly the other player) 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, which is connected to a virtual slider since no real dials or sliders are connected to the current system. Whether or not this slider is part of the default application, i.e. whether it is visible when first entering the virtual environment, is also easily changed.

6. Conclusion

The system described in this paper provides a simple and powerful way in which to construct and modify immersive virtual environments. Care has been taken not to provide a monolithic system that is a completely general programming system which would generate very complex and hard to modify dataflow diagrams, but rather to provide filters and behaviours useful to immersive environment creators. Inevitably there will be applications requiring the coding of new behaviours but the effort required will small given the infrastructure that already exists.

7. Acknowledgements

Thanks to Martin Usoh for technical help on the Provision. This work is funded in part by the EPSRC/ROPA grant GR/K38090, The Distributed Extensible Virtual Reality Laboratory. Anthony Steed is supported by a UK EPSRC award.

References

[1] Division. *dVISE for Unix Workstations, User Manual (draft)*, 1995.

[2] E. 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.

[3] E. Gobbetti and J.-F. Balaguer. An integrated environment to visually construct 3d animations. *Computer Graphics, Proceedings of SIGGRAPH 95*, pages 395–398, 1995.

[4] R. S. Kalawsky. *The Science of Virtual Reality and Virtual Environments*. Addison-Wesley, 1993.

[5] W. Lytle. Vpla: Visual programming language for animation. Technical Sketch, SIGGRAPH95, 1995.

[6] M. Najork. *Programming in Three Dimensions*. PhD thesis, University of Illinois at Urbana-Chapaign, 1994.

[7] W. Sherman. Integrating virtual environments into the dataflow paradigm. In *Fourth Eurographics Workshop on ViSC, Abingdon, UK*, April 1993.

[8] 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.

[9] A. Steed and M. Slater. 3d interaction with the desktop bat. *Computer Graphics Forum*, 14(2), 1995.

[10] 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.