A 'plug and play' approach to testing virtual environment interaction techniques

James S. Willans and Michael D. Harrison

Human-Computer Interaction Group
Department of Computer Science, University of York
Heslington, York YO10 5DD, U.K.
e-mail: {James.Willans,Michael.Harrison}@cs.york.ac.uk

Abstract. It is generally agreed that the usability of virtual environment interaction techniques is very poor. One reason for this is because toolkits used by virtual environment developers supply a small number of predefined techniques which are expected to be used regardless of context. In addition, there is no software to facilitate the design and testing of interaction techniques akin to that found for the appearance of the environment. We have developed the Marigold toolset to aid in the systematic design, testing and refining of virtual environment interaction techniques. The toolset uses a visual hybrid specification as a starting point. In this paper we demonstrate how Marigold can be used to aid in determining the suitability of interaction techniques by the rapid testing of alternatives in a 'plug and play' style.

1 Introduction

Much of the work concerning virtual environments has addressed technological issues such as the development of new toolkits and methods for distributing the computational load inherent in such systems. However, the usability of virtual environments is generally poor largely because of the interaction techniques used within such environments [3]. This situation can be attributed to two main reasons. Firstly, popular virtual environment toolkits such as Superscape [19] and dVise [6] provide a small number of predefined techniques (bound to physical devices) which are expected to be used regardless of context. Secondly, while there is much software available which facilitates the design and prototyping of the environment's appearance (for example 3DStudio [5]), the corresponding facilities for interaction techniques do not exist.

Stanney notes that 'if humans cannot perform effectively within virtual environments, then further pursuit of the technology may be fruitless' [17]. If virtual environments *are* to be useful in a wider context, it is important to ensure their usability by carefully determining appropriate interaction techniques for the individual requirements of the environment. For instance, consider the following scenario: a technique is required for an environment that allows a potential house buyer to explore a residential area. There are many interaction techniques which will support such navigation, but the important question is which technique is

suitable in this context? Such questions are difficult to answer with any degree of certainty using abstract designs. This is because the diversity of virtual environments means that each environment has a large number of unique factors that need to be considered when designing the interaction technique. For the same reason, guidelines for designing interaction techniques, such as those presented in [2], can only give an outline indication of the design. We suggest that an exploratory approach is more desirable, where the developer can design techniques and test these with users using a prototype of the whole environment. The advantages of this are that the interaction technique is validated in the context of all the elements of the environment (world objects and viewpoints, for instance) and users are involved within the design process.

In this paper we present the Marigold toolset which supports a rapid transition between the design and testing of virtual environment interaction techniques. This process begins with a visual specification of the desired interaction technique which is then 'plugged' into a virtual environment. This is described in section 2. In section 3 we discuss and demonstrate how the process aids rapid exploration of alternative techniques. Section 4 examines related work. Finally, in section 5 we present our conclusions.

2 Marigold

We have developed the Marigold toolset to aid in the rapid designing, testing and refining of virtual environment interaction techniques. The toolset is designed to support a similar process to the Statemate tool [8] (based around the statechart formalism [7]) that provides a means of visually specifying the behaviour of dynamic systems. This behaviour can then be explored interactively. Although Marigold currently only supports navigation techniques, we are extending it to support selection and manipulation techniques.

Marigold consists of two tools. The interaction technique builder (ITB) supports visual specification of an interaction technique using the notation presented in [15, 16]. From the ITB a stub of the interaction technique is generated. This stub is an environment independent description of the interaction technique. By this, we mean that it does not make commitments to the inputs and outputs of the technique. The second tool, the prototype builder (PB), provides a visual method of 'plugging' the generated stubs of interaction techniques into the other elements of the virtual environment (e.g. the devices and visual renderings). The code for the virtual environment is automatically generated from the PB.

This approach is rapid for a number of reasons. Firstly, the concurrency inherent in virtual environment interaction techniques is captured in the notation. Consequently, the developer can easily and accurately describe concurrent behaviour. Secondly, both tools verify automatically that the specifications are semantically correct, hence there is very little chance that the automatically generated code will fail to produce any results. Finally, as the interaction techniques are defined in an environment independent form, it is very easy for developers

to 'plug and play' interaction techniques to determine their suitability within varying environment configurations.

2.1 The specification formalism and interaction technique builder

Virtual environments can be thought of as consisting of a hybrid of continuous and discrete components [10]. The hybrid specification utilised by the ITB was specifically developed for the specification of virtual environment interaction techniques and is described in [15]. The notation uses Petri-nets [14] to describe the discrete behaviour and a newly developed extension to describe the continuous behaviour. Like Statecharts [7], the formalism is not concerned with the implementation of the states, only the behavioural ordering (what can happen and in what sequence). This reduces complexity and allows insight into requirements on the design rather than a premature focus on implementation.

We will describe the specification formalism by way of an example. The mouse based flying interaction technique enables flying through a virtual environment using the desktop mouse. Variations of this technique are used in many desktop virtual environment packages (e.g. the virtual production planner [1] and VRML [4]). One variation works as follows. The technique is initiated by pressing of the middle mouse button and moving the mouse away from the clicked position. The user's speed and direction is directly proportional to the angle and distance between the current pointer position at the point the middle mouse button was pressed. Flying is deactivated by a second press of the middle mouse button.

The hybrid specification of the mouse based flying interaction technique is shown in figure 1 within Marigold ITB[1]. The technique has one input: *mouse*, and one output: *position*. When the middle mouse button is pressed the *middle m/butt* sensor is activated and the *start* transition fired (1). The *start* transition enables the continuous flow which updates *origin* with the current mouse position (2) (taken from the *mouse* plug). A token is then placed in the *idle* state. When the *out origin* sensor detects that the mouse has moved away from the *origin* position, transition (3) is fired which moves the token from the *idle* state to the *flying* state. A token in the *flying* state enables the continuous flow which calculates the translation on *position* using the current mouse position and the *origin* (4). This is then continuously outputted to the the *position* plug. Whenever the *flying* state is enabled, the inhibitor connecting this state to the *start* transition implies that the *start* transition cannot be re-fired. When the *at origin* sensor detects that the mouse has moved back to the *origin* position, a transition is fired which returns the token from the *flying* state to the *idle* state closing the continuous flow and halting the transformation on *position*. Regardless of whether the technique is in the *idle* or the *flying* state, the technique can be exited by the *middle m/butt* sensor becoming true and firing either one of the two *exit* transitions (5 or 6).

[1] The numbers are not part of the specification and have been annotated to the screenshot.

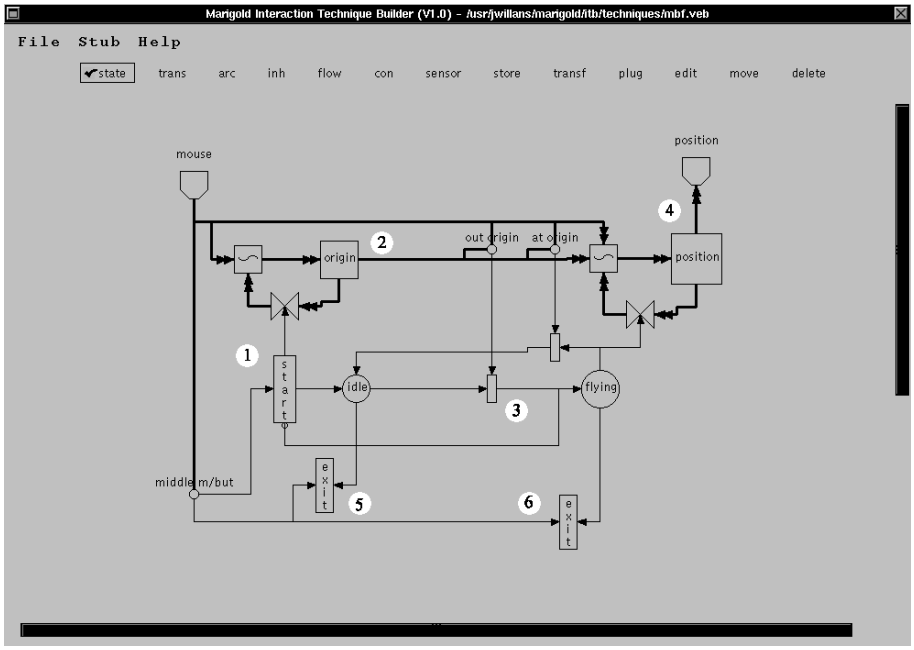

Fig. 1. The mouse based flying hybrid specification within the ITB

The toolbar at the top of the diagram contains an option for each of the node and link components constituting the specification. The ITB enforces the semantics of the specification and only allows legal connections between components. Additionally, the tool also tries to maintain clarity of specification by intelligent routing of the visual connections between the components.

There are two stages to the refinement of an interaction technique specification to a prototype. The first stage takes place in the ITB and involves adding a small amount of C/Maverik (virtual environment library [9]) code to some of the nodes within the specification. There are three types of code that can be added, we will describe these in the context of the mouse based flying example:

- variable code - this is placed in the plugs of the specification (▷). It describes what kind of information flows in and out of the plugs and, hence, around the specification. Illustrated in figure 2 (a) is the code added to the *mouse* plug. An integer variable represents the state of the mouse buttons and a vector represents the mouse position.
- conditional code - this is placed in some transitions (▯) and all sensors (⊸). It describes what state the data flowing into that component must adhere to in order for the component to fire. Illustrated in figure 2 (b) is the code added to the *middle m/but* sensor. The ITB informs the developer which data flows in and out of the node (i.e. what data they are able to access

within their code). This code specifies that when the middle mouse button is pressed a boolean value is returned.

– process code - this is placed in all transformers ( ) and denotes how the information flowing into the transformer is transformed when enabled. Illustrated in figure 2 (c) is the code added to the *position* transformer. This describes how position should be transformed using the current mouse position and the origin position.

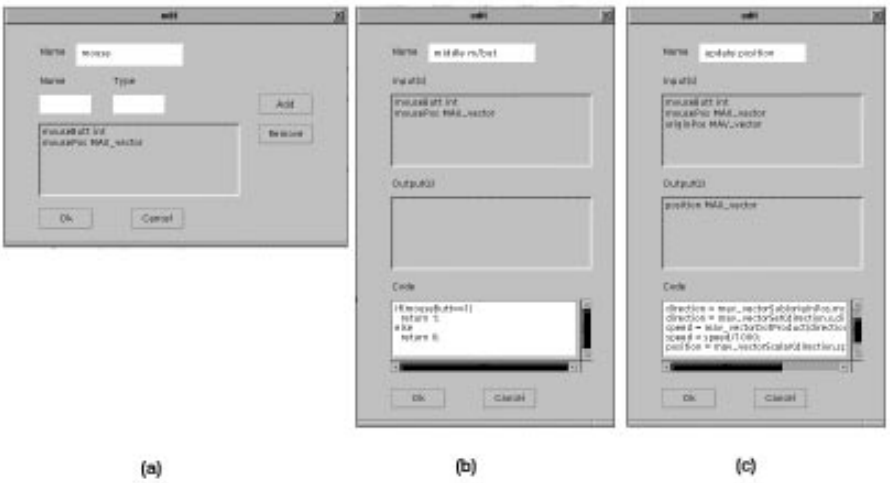

Fig. 2. a) Adding variables to the mouse input plug b) Adding conditional code to the middle mouse button sensor c) Adding process code to the position transformer

2.2 The prototype builder

The Marigold PB provides a method of visually 'plugging' an interaction technique stub generated from the ITB with the other elements of the virtual environment such as devices, viewpoints and visual renderings. Shown in figure 3 is the mouse based flying interaction technique example within the PB integrated into an environment. As can be seen in this illustration, each node has a set of variables, the variables for the mouse based flying technique (mbf) are those placed in the techniques plugs within the ITB. What cannot be seen, from this black and white figure, is that each variable has a background colour denoting whether it an input or output variable. The relation between the environment elements is expressed by wiring these variables together using transitions. The tool automatically verifies that the variables being joined are the correct type.

Once the specification is complete, the code for the virtual environment (in its entirety) can be automatically generated.

Within the mouse based flying example (figure 3), we have linked a *desktop mouse* as an input to the technique and a *viewpoint* as an output from the technique. Additionally, we have inserted a number of world object visual renderings. However, since these remain static during interaction, they are not linked to any other elements of the environment.

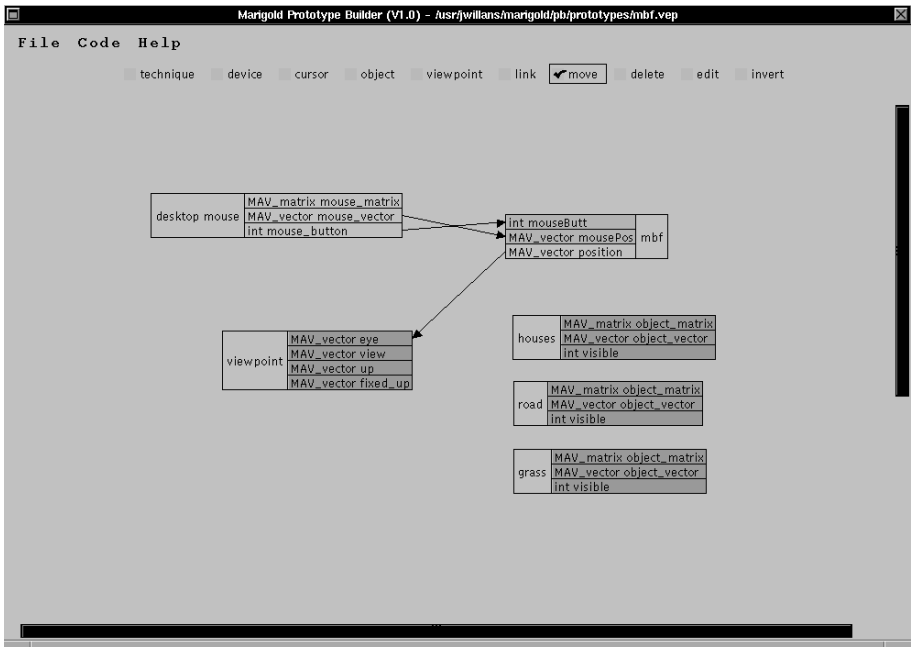

Fig. 3. The prototype specification for mouse based flying within the PB

3 Discussion

There are two main values to the approach introduced in section 2. Firstly, the hybrid specification can be verified for desirable usability properties. This is out of the scope of this paper and is discussed in [20]. Secondly, interaction techniques can be rapidly built and tested. To exemplify this, the mouse-based flying example discussed in the section 2 was inserted into an environment for the navigation of a hypothetical planned housing estate. Consider a converse possibility of plugging the two handed flying interaction technique [12] into this example. The existing interaction technique is deleted from the PB specification and the new technique stub added and wired, where necessary, to the other

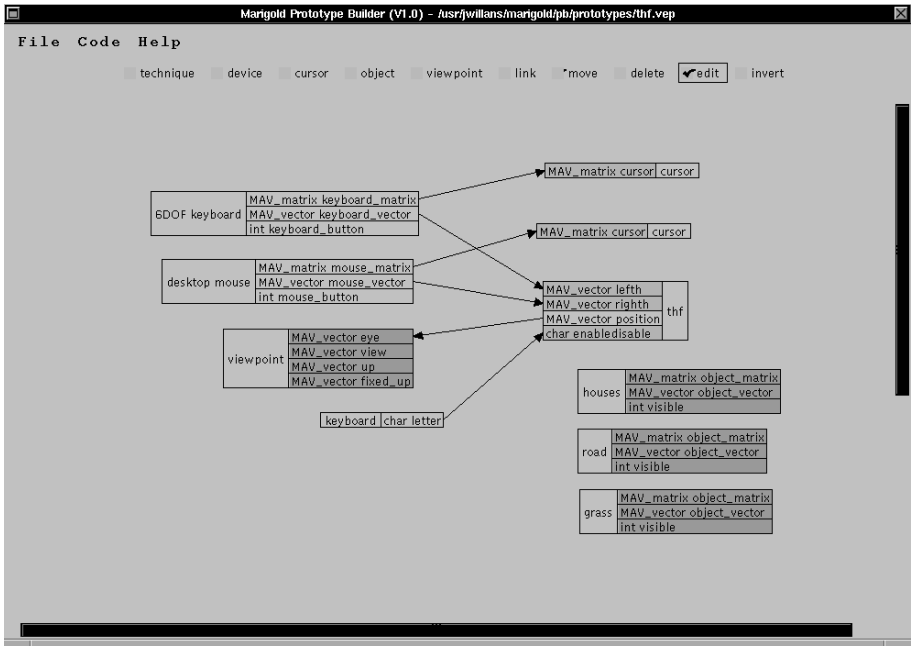

Fig. 4. The two handed flying interaction technique prototype specification

nodes. Illustrated in figure 4 is the PB with the same visual renderings and viewpoints as the previous example (shown in figure 3), however wired to the new interaction technique [12].

The two handed flying interaction technique, detailed in [12], enables flying through a virtual environment in a direction determined by the vector between the user's two hands and at a speed relative to their hand separation. Movement can be halted as the user brings their hands together. The principal idea behind the technique is to utilise user proprioception.

In addition, the device inputs within the environment have changed. Although the two handed flying technique is designed for use with two 6 degrees of freedom trackers (Polhemus magnetic trackers, for instance), we have substituted two pseudo devices into the technique so that it can be tested using the devices available on a desktop workstation (the keyboard controlling the left hand and the mouse controlling the right hand). As it is not possible to feel the relative position of these devices proprioceptively, they are also mapped onto cursors within the environment so that their relative positioning can be perceived visually within the viewpoint. The additional keyboard mapping (*keyboard*) activates and deactivates the technique. Such pseudo devices configurations provide a means for developers to 'play' and test whether the technique is semantically correct. However, these would be substituted for physical devices to allow users to 'play' so that the usability of the technique can be evaluated.

Such 'plug and play' substitution is very simple using the PB and substantially simpler, clearer and faster than experimentation by direct alteration of program code. Although it is difficult to appreciate the behaviour of a virtual environment from a static image, the screenshot of the environment generated from the prototype specification shown in figure 4 is illustrated in figure 5.

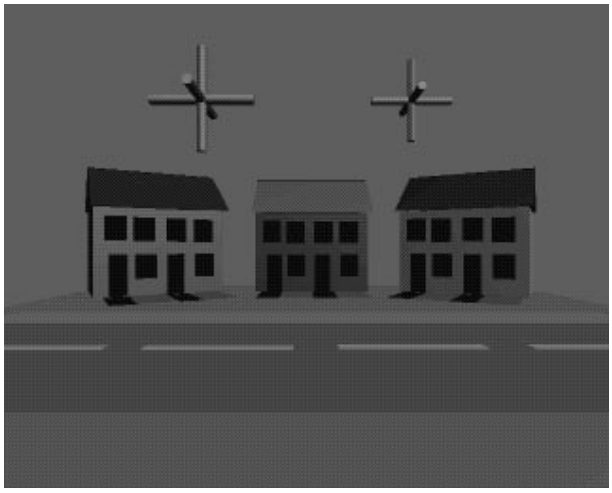

Fig. 5. The virtual environment as generated from Marigold PB using the two handed flying interaction technique

4 Related work

A method for designing interaction techniques is presented in [2]. The main motivation behind this work is to provide a method of fitting interaction techniques to the needs of the task and a framework is presented which supports this process. The framework results in a high level description of the components constituting the required interaction technique(s). The work presented here complements this process because it is able to refine these high level descriptions to prototypes and support experimental verification that the designed technique is indeed correct.

A number of methods have been developed which relate visual specifications to the implementation of virtual environment behaviour. In [11, 13] Jacob et al. present a user interface management system (UIMS) which links a visual hybrid notation to the behaviour of a virtual environment. Hence, changes made in the specification are propagated to the implementation. Our work differs from this in that they have an implementation as a starting point which is linked to the higher level notation (in the traditional UIMS manner), however we refine from the specification to the prototype. Similarly, the work presented in [18] provides

a method of visually constructing the behaviour of virtual environments while immersed within the environment. The visual notation provides a very high level set of components which are plugged together to form a specification. However, the high-level nature of the components severely limits what can be achieved.

5 Conclusions

In this paper we have identified that the careful design and selection of virtual environment interaction techniques is a key factor in developing usable virtual environments. We have presented a tool supported approach, motivated by an existing process, which provides the means to rapidly and systematically design, test and refine interaction technique. We have demonstrated how this 'plug and play' style of development can support the rapid exploration of alternative interaction techniques.

Acknowledgements

We are grateful to Jon Cook at the Advanced Interface Group at the University of Manchester for his help with the details of Maverik. We are also grateful to Shamus Smith and David Duke for their comments on this work.

References

1. BBC/Colt International. Virtual production planner, 1997.
2. D.A. Bowman. *Interaction Techniques for Common Tasks in Immersive VirtualEnvironments - Design, Evaluation and Application.* PhD thesis, Georgia Institute of Technology, 1999.
3. D.A. Bowman and L.F. Hodges. User interface constraints for immersive virtual environment applications. Technical report, Graphics, Visualisation and Usability Center, Georgia Institue of Technology, 1995.
4. Rikk Carey and Gavin Bell. *The Annotated VRML 2.0 Reference Manual.* Developers Press, 1997.
5. Autodesk corporation. 3DStudio. 111 McInnis Parkway, San Rafael, California, 94903, USA.
6. Pierre duPont. Building complex virtual worlds without programming. In Remco C. Veltkamp, editor, *Eurographics'95*, pages 61–70, 1995.
7. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.
8. David Harel, Hagi Lachover, Amnon Naaad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–413, July 1990.
9. Roger J. Hubbold, Xiao Dongbo, and Simon Gibson. Maverik - the Manchester virtual environment interface kernel. In Martin Goebel and Jacques David, editors, *Proceedings of 3rd Eurographics Workshop on Virtual Environments.* SpringerVerlag, 1996.

10. Robert J.K. Jacob. Specifying non-WIMP interfaces. In *CHI'95 Workshop on the Formal Specification of User Interfaces Position Papers*, 1995.
11. Robert J.K. Jacob. A visual language for non-wimp user interfaces. *Proceedings IEEE Symposium on Visual Languages*, pages 231–238, 1996.
12. Mark R. Mine, Fredrick P. Brook Jr, and Caro H.Sequin. Moving objects in space: Exploiting proprioception in virtual-environment interaction. In *SIGGRAPH 97*, 1997.
13. S.A. Morrison and R.J.K. Jacob. A specification paradigm for design and implementation of non-wimp human-computer interaction. In *ACM CHI'98 Human Factors in Computing Systems Conference*, pages 357–358. Addison-Wesley/ACM Press, 1998.
14. Wolfgang Reisig. *Petri Nets.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1982.
15. Shamus Smith and David Duke. Virtual environments as hybrid systems. In *Eurographics UK 17th Annual Conference*, 1999.
16. Shamus Smith, David Duke, and Mieke Massink. The hybrid world of virtual environments. *Computer Graphics Forum*, 18(3):C297–C307, 1999.
17. Kay M. Stanney, Ronald R. Mourant, and Robert S. Kennedy. Human factors issues in virtual environments. *Presence*, 7(4):327–351, August 1998.
18. Anthony J. Steed. *Defining Interaction within Immersive Virtual Environments.* PhD thesis, Queen Mary and Westfield College, 1996.
19. Superscape Corporation. Superscape, 1999. 3945 Freedom Circle, Suite 1050, Santa Clara, CA 95054, USA.
20. James S. Willans and Michael D. Harrison. A toolset supported approach for designing and testing virtual environment interaction techniques. *International Journal of Human-Computer Studies (submitted)*, 1999.