1

Virtuality Builder II On the Topic of 3D Interaction

Jean-Francis Balaguer
Enrico Gobbetti

Computer Graphics Laboratory
Swiss Federal Institute of Technology, Lausanne

1.1 INTRODUCTION

Most of today's user interfaces for 3D graphics systems still predominantly use 2D widgets, even though current graphical hardware should make it possible to create applications in which the user directly manipulates aspects of three-dimensional synthetic worlds. The difficulties associated with achieving the key goal of immersion has led the research in virtual environments to concentrate far more on the development of new input and display devices than on higher-level techniques for 3D interaction.

It is only recently that interaction with synthetic worlds has tried to go beyond straightforward interpretation of physical device data (NSF, 1992), (Balaguer and Mangili, 1992). The design space for 3D interaction tools and techniques remains mostly unexplored, while being far larger than in standard 2D applications. Moreover, as stated by Myers, "the only reliable way to generate quality interfaces is to test prototypes with users and modify the design based on their comments" (Myers, 1989). The creation of complex interactive applications is an inherently iterative process that requires user interface tools, such as toolkits or frameworks.

The lack of experience in 3D interfaces makes it extremely difficult to design 3D interface toolkits or frameworks. We believe that offering the possibility to rapidly prototype and test novel interaction techniques should be the primary goal of such tools. It is therefore more important for these tools to provide a wide range of interaction components, than to enforce a particular interface style.

In this paper we present the *Virtuality Builder II* (*VB2*) framework developed at the Swiss Federal Institute of Technology for the construction of 3D interactive applications. First, we'll give an overview of the design concepts of *VB2*. Next, we'll concentrate on how users interact with dynamic models through direct manipulation, gestures, and virtual tools. More details on the rendering and modeling clusters are found in (Gobbetti et al., 1993), and more detailed explanations of the dependency maintenance algorithms, as well as on their use to implement tools behavior, are found in (Gobbetti, Balaguer et al., 1993).

1.2 DESIGN CONCEPTS

VB2 is an object-oriented framework designed to allow rapid construction of applications using a variety of 3D devices and interaction techniques. As shown in figure 1, *VB2* applications are composed of a group of processes communicating through inter-process communication (*IPC*). A central process manages the model of the virtual world, and simulates its evolution in response to events in the form of *IPC* messages coming from the processes that encapsulate asynchronous input devices. Sensory feedback to the user can by provided by several output devices. Visual feedback is provided by real-time rendering on graphics workstations, while audio feedback is provided by *MIDI* output and playback of prerecorded sounds.

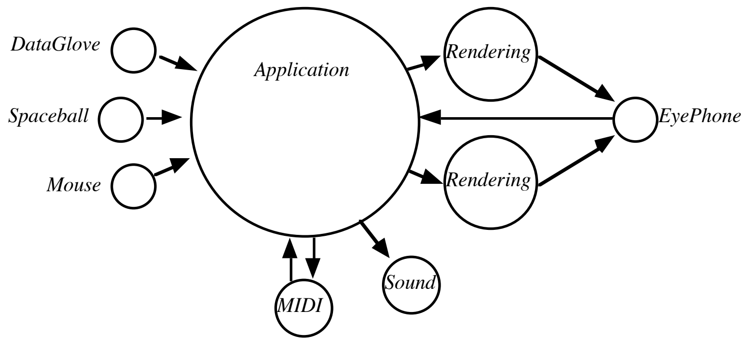


Figure 1. Overall structure of *VB2*

In order to obtain animated and interactive behavior, the system has to update its state in response to changes initiated by sensors attached to asynchronous input devices such as timers or trackers. The virtual world can be seen as a network of interrelated objects whose behavior is specified by the actions taken in response to changes in the objects on which they depend.

To model this kind of behavior, three different aspects have to be considered:

- the *state* of the system;
- the *long-lived relations* that have to be maintained between the different components of the state;
- the *sequencing relations* between states.

In *VB2*, each one of these aspects is modeled using different primitive elements: *active variables* are used to store the state of the system, *reactions* to maintain object's properties, *hierarchical constraints* to declaratively represent long-lived multi-way relations between active variables, and *daemons* to react to variable changes for imperatively changing between different system states. A central state manager is responsible for adding, removing, and maintaining all active constraints as well as managing the system time and activating reactions and daemons. This way, imperative and declarative programming techniques can be freely mixed to model each aspect of the system with the most appropriate means, much as in the programming language *Kaleidoscope* (Freeman-Benson, 1990).

1.2.1 Information modules

All *VB2* objects are instances of classes in which dynamically changing information is defined with active variables related through hierarchical constraints. Grouping active variables and constraints in classes permits the definition of information modules that provide levels of abstraction that can be composed to build more sophisticated behavior.

1.2.2 Active variables

Active variables are the primitive elements used to store the system state. An active variable maintains its value and keeps track of its state changes. Upon request, an active variable can also maintain the history of its past values. A variable's history can be accessed using the variable's local time, which is incremented at each variable's state change, or using the system's global time, which is incremented at each atomic constraint operation. This simple model makes it possible to elegantly express time-dependent behavior by creating constraints or daemons that refer to past values of active variables.

1.2.3 Reactions and Transactions

In VB2, modifying some active variables of an information module requires that a *transaction* on this module has been opened. Transactions are used to group changes on active variables of a same module. *Reactions* register themselves with a set of active variables and are activated at the end of a transaction. They are used to enforce object invariants as well to maintain any kind of relation between a set of active variables. The reaction code is imperative and may result in the opening of new transactions on other modules as well as in the invalidation of the value of modified variables. All the operations performed during a transaction are considered as occurring within the same time slice.

1.2.4 Hierarchical Constraints

Multi-way relations between active variables are specified in *VB2* through *hierarchical constraints*, introduced in *ThingLab* (Borning et al., 1987, 1989) for the construction of two-dimensional user interfaces.

Constraint objects are composed of a declarative part, which defines the type of relation that has to be maintained, together with set of concerned active variables, and an imperative part, which is a list of possible methods that could be used to maintain the constraint. Constraint variables are located either directly or through *symbolic paths.* A symbolic path is an indirect reference to a variable described by the sequence of names of the active variables that have to be traversed to reach the referenced variable. Constraint methods are general procedures of any complexity that ensure the satisfaction of the constraint after their execution by computing certain of the constrained variables as a function of some of the others. A priority level is associated with each constraint to define the order in which constraints need to be satisfied in case of conflicts: this way, both required and preferred constraints can be defined for the same active variable.

A central constraint solver is activated each time a constraint is added to the graph or removed from it, and each time an active variable changes its value. Its goal is to maintain symbolic paths, and to decide which constraints should be satisfied, which method should be used for each constraint, and in what order these methods should be invoked. All the operations that the constraint manager performs to address these needs are considered as occurring at the same time and do not modify the system time.

We based our solver on the *DeltaBlue* algorithm (Freeman-Benson and Maloney, 1989), which we extended to perform lazy evaluation and deal with constraints composed of methods having multiple outputs. Constraints using symbolic paths are handled by transforming them to fixed reference constraints that are automatically removed from the network and reconnected to the correct variables each time a component of a symbolic path changes, as in the user-interface toolkit *Multi-Garnet* (Sannella and Borning, 1992).

1.2.5 Daemons

Daemons are the imperative portion of *VB2*. They are the objects which permit to define the sequencing between system states. Daemons register themselves with a set of active variables and are activated each time their value changes. The action taken by a daemon can be a procedure of any complexity that may create new objects, perform input/output operations, change active variables' values, manipulate the constraint graph, or activate and deactivate other daemons. The execution of a daemon's action is sequential and each manipulation of the constraint graph advances the global system time. A priority level is associated which each daemon to define the activation order.

1.3 INTERACTION TECHNIQUES

In most typical interactive applications, users spend a large part of their time entering information, and several types of input devices, such as 3D mice and *DataGloves*, are used to let them interact with the virtual world. Using these devices, the user has to provide at high speed a complex flow of information, and a mapping between the information coming from the device sensors and the actions in the virtual world has to be devised.

The definition of this mapping is crucial for interactive applications, because it defines the way users communicate with the computer. Ideally, interactive 3D systems should allow users to interact with synthetic worlds in the same way they interact with the real world, thus making the interaction task more natural and reducing training.

1.3.1 Direct manipulation

In most systems, the interaction mapping is hard coded and directly dependent on the physical structure of the device used (for example, by associating different actions to the various mouse buttons). This kind of behavior is obtained in *VB2* by attaching constraints directly relating the sensors' active variables to variables in the dynamic model, as in the example of figure 2. These constraints define the interaction metaphor, and their activation and deactivation are triggered by daemons.

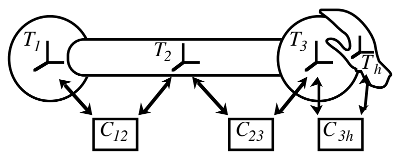


Figure 2. Graphical objects grabbed by user with constraints

Such a direct mapping between the device and the dynamic model is straightforward for tasks where the relations between the user's motions and the desired effect in the virtual world is mostly physical, as in the example of grabbing an object and moving it, but needs to be very carefully thought out for tasks where user's motions are intended to carry out a meaning. In this latter case, hardwiring virtual world actions to specific sensor values forces commitments that risk reducing device expressiveness and can make applications difficult to use (Fels and Hinton, 1990).

In order to overcome these problems, *mediator* objects can be interposed between sensors and models to transform the information accordingly to interaction metaphors. Two major types of mediators are used in VB2:

- *adaptive pattern recognizers*, to enhance sensor data with classification information, hence increasing the expressive power of the input devices;
- *virtual tools*, encapsulations of visual appearance and behavior, to present selective views of models' information and offer the interaction metaphors to control it.

Information transformation is obtained by propagation through the mediators' internal constraint networks. Multiple mediators can be simultaneously active to allow manipulation of several models at the same time or of a single model with different interaction metaphors.

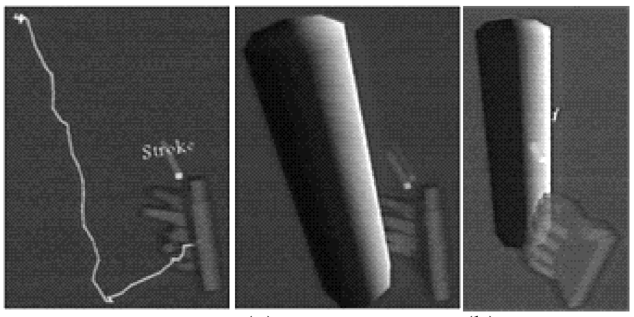
1.3.2 Hand gestures

VB2 uses a gesture recognition system linked to the *DataGlove*. The gesture recognition system has to classify, on the basis of previously seen examples, movements and configurations of the hand in different categories. Once the gesture is classified, parametric information for that gesture can be extracted from the way it was performed, and an action in the virtual world can be executed. This way, with a single gesture, both categorical and parametric information can be provided at the same time in a natural way (Rubine, 1991). A visual and an audio feedback on the type of gesture recognized and on the actions executed are usually provided in *VB2* applications to help the user understand system's behavior.

VB2's gesture recognition is subdivided into two main portions: posture recognition, and path recognition. The type of gesture chosen is compatible with Buxton's suggestion (Buxton, 1986)(Buxton, 1990) of using physical tension as a natural criterion for segmenting primitive interactions: the user, starting from a relaxed state, begins a primitive interaction by tensing some muscles and raising its state of attentiveness, performs the interaction, and then relaxes the muscles. In our case, the beginning of an interaction is indicated by positioning the hand in a recognizable posture, and the end of the interaction by relaxing the fingers.

The posture recognition subsystem is continuously running and is responsible for classifying the user's hand finger configurations. Once a configuration has been recognized, the hand data is accumulated as long as the hand remains in the same posture. This data is then passed to the path recognition subsystem to classify the path. A gesture is therefore defined as the path of the hand while the hand fingers remain stable in a recognized posture.

The gesture recognition system is a way to enhance the data coming from the sensors with classification information and thus provides an augmented interface to the device. The ability to specify the mapping through examples makes applications easier to adapt to the preferences of new users, and therefore makes them simpler to use.



(a) (b) (c)

Figure 3a, 3b. Creating a cylinder by gestural input
Figure 3c. Grabbing the cylinder through posture recognition

1.3.3 Virtual tools

The amount of information that can be controlled on a three-dimensional object and the ways that could be used to control it are enormous. Gestural input techniques and direct manipulation on the objects themselves offer only partial solutions to the interaction problem, because these techniques imply that the user knows what can be manipulated on an object and how to do it. The system can guide the user to understand a model's behavior and interaction metaphors by using mediator objects that present a selective view of the model's information and offer the interaction metaphor to control this information. We call these objects *virtual tools*.

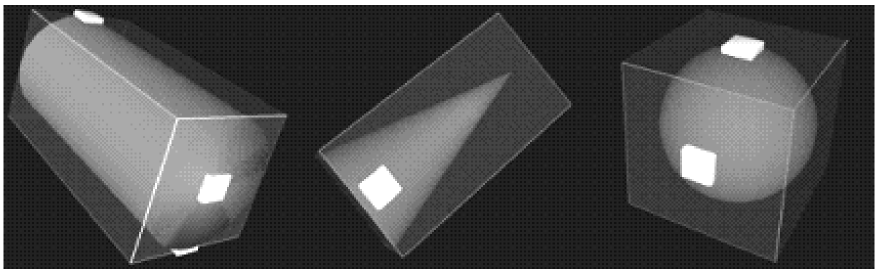


Figure 4. Examples of simple virtual tools

VB2's virtual tools are first class objects, like the widgets of *UGA* (Conner et al., 1992), that encapsulate a visual appearance and a behavior to control and display information about application objects.

The visual appearance of a tool must provide information about its behavior and offer semantic feedback to the user during manipulation. In *VB2*, the visual appearance of a tool is described using a modeling hierarchy. In fact, most of our tools are defined as articulated structures that can be manipulated using inverse kinematics techniques, as tools can often be associated with mechanical systems.

The tool's behavior must ensure the consistency between its visual appearance and the information about the model being manipulated, as well as allow information editing through a physical metaphor. In *VB2*, the tool's behavior is defined as an internal constraint network, while the information required to perform the manipulation is represented by a set of active variables.

In *VB2*, virtual tools are fully part of the synthetic environment. As in the real world, the user configures its workspace by selecting tools, positioning and orienting them in space, and binding them to the models he intends to manipulate. Multiple tools may be attached to a single model in order to simultaneously manipulate different parts of the model's information, or the same parts using multiple interaction metaphors.

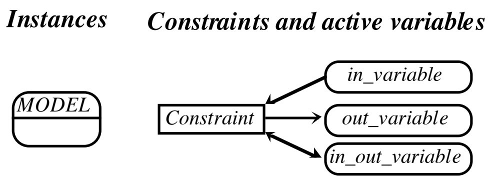


Figure 5. Notation

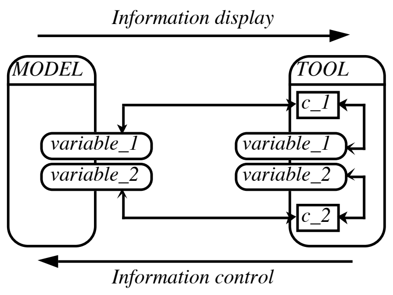


Figure 6. Model and virtual tool

1.3.3.1 Virtual tool protocol

The user declares the desire to manipulate an object with a tool by *binding* a model to a tool. When a tool is bound, the user can manipulate the model using it, until he decides to *unbind* it.

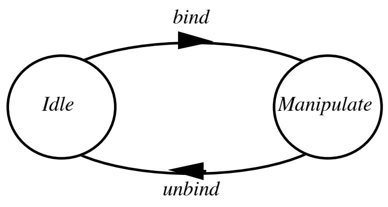
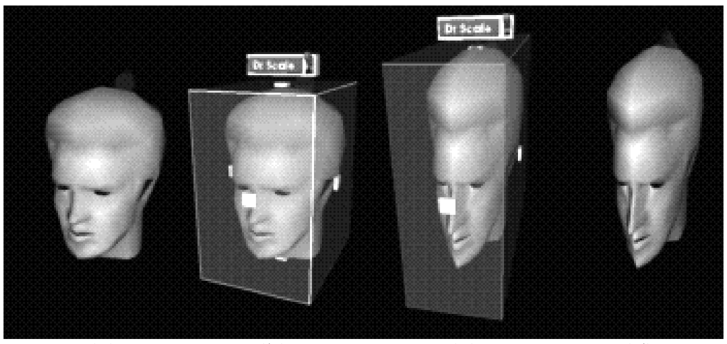


Figure 7. Tool's state transitions

When a *bind* message is sent to a tool, the tool must first determine if it can manipulate the given model, identifying on the model the set of public active variables requested to activate its binding constraints. Once the binding constraints are activated, the model is ready to be manipulated. The binding constraints being generally bi-directional, the tool is always forced to reflect the information present in the model even if it is modified by other objects.

When a tool is bound to a model, the user can manipulate the model's information through a physical metaphor. This iterative process composed of elementary manipulations is started by the selection of some part of the tool by the user, resulting in the activation of some constraint like, for example, a motion control constraint between the 3D cursor and the selected part. User input motion results in changes to the model's information through propagation of device sensor values through the tool's constraint network, and so until the user completes the manipulation, deselecting the tool's part. Gestural input techniques can be used to initiate and control a tool's manipulations, for example by associating selection and deselection operations to specific hand postures.

The *unbind* message is sent to a tool to detach it from the object it controls. The effect is to deactivate the binding constraints in order to suppress dependencies between the active variables of the tool and model. Once the model is unbound, further manipulation of the tool will have no effect on the model.



(a). (b) (c) (d)

Figure 8a. Model before manipulation
Figure 8b. A scale tool is made visible and bound to the model
Figure 8c. The model is manipulated via the scale tool
Figure 8d. The scale tool is unbound and made invisible

1.3.3.2 Composition of virtual tools

Since virtual tools are first class dynamic objects in *VB2*, they can be assembled into more complex tools much in the same way as simple tools are built on top of a modeling hierarchy. The reuse of abstractions provided by this solution is far more important than the more obvious reuse of code.

An example of a composite tool is *Dr. Map*, which is a virtual tool used to edit the texture mapping function of a model by controlling the parallel projection of an image on the surface of the manipulated model. The tool is defined as a plane on top of which is mapped the texture and a small arrow icon displays the direction of projection. In order to compute the mapping function to be applied to the model, the tool needs to know the texture to be used, the position and orientation of the model in space, and the position and orientation of the tool in space. The textured plane represents the image being mapped, and a *Dr. Plane* tool allows manipulation of the plane in order to change the aspect ratio of the texture's image. Pressing the grabber button allows the user to position and orient the tool in the 3D space, hence specifying the direction and origin of the texture projection.

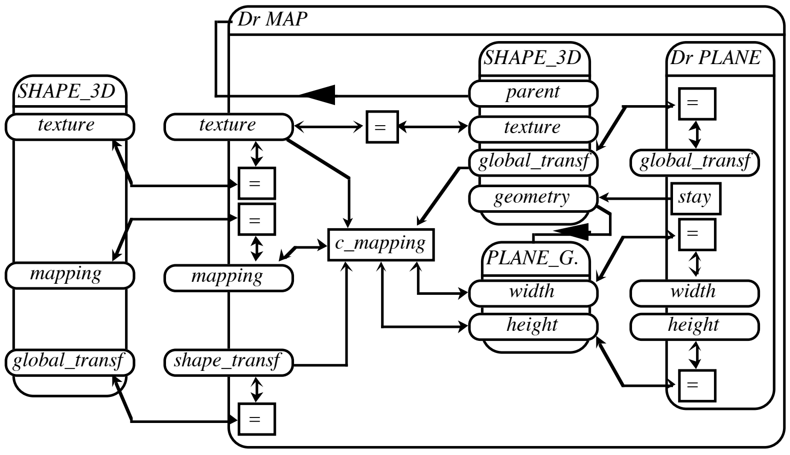
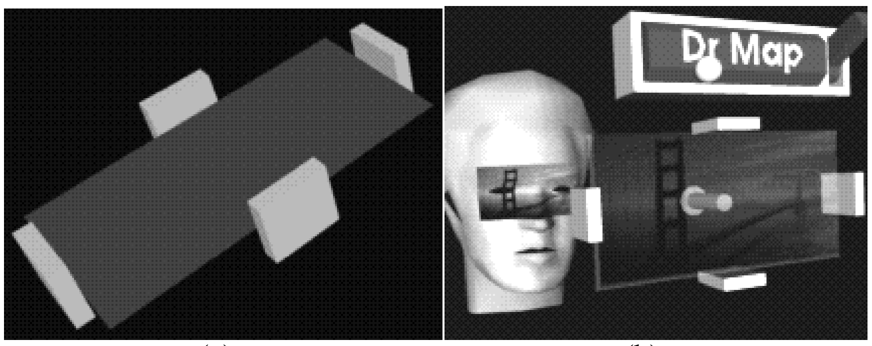


Figure 9. *Dr. Map*'s simplified constraint network.



(a) (b)

Figure 10a. View of *Dr. Plane*
Figure 10b. View of *Dr. Map*

Similarly, the material editing tool is built out of color tools and the light tool is built out of a cone tool. By reusing other tools we enforce consistency of the interface over all the system, allowing users to perceive rapidly the actions they can perform. Building tools by composing the behavior and appearance of simpler objects is relatively easy in *VB2*: for example, *Dr. Map* tool was built and tested by one person in about a couple of hours. The fast prototyping abilities of the system are very important for a framework aimed at experimenting with 3D interaction.

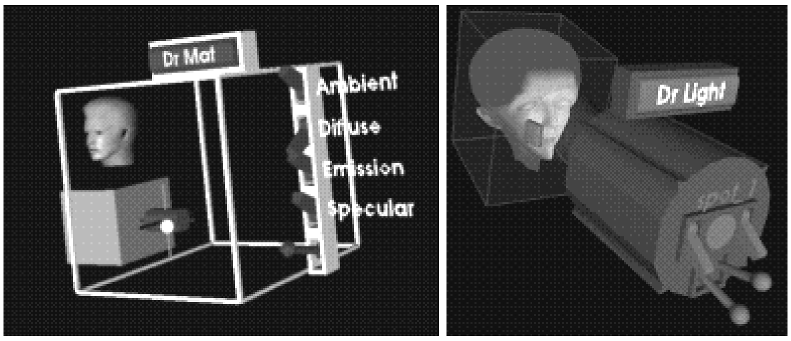


Figure 11. View of some other tools

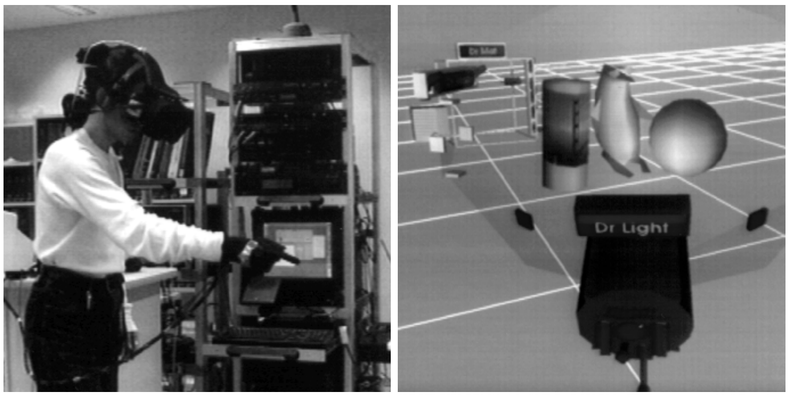


Figure 13 Synthetic environment

1.4. CONCLUSIONS AND FURTHER WORK

We have presented the *VB2* framework for the construction of three-dimensional interactive applications. In *VB2*, multiple devices can be used to interact with the synthetic world through various interaction paradigms. *VB2* is implemented in the object oriented language *Eiffel* (Meyer, 1992) on *Silicon Graphics* workstations, and is currently composed of over 300 classes.

Interaction techniques range from direct manipulation to gestural input and three-dimensional virtual tools. Adaptive pattern recognition is used to increase input device expressiveness by enhancing sensor data with classification information. Tools, which are encapsulations of visual appearance and behavior, present a selective view of the manipulated model's information and offer the interaction metaphor to control it. Since tools are first class objects, they can be assembled into more complex tools. much in the same way simple tools are built on top of a modeling hierarchy. New three-dimensional tools are easily added to the system, and their number is rapidly growing.

Hierarchical constraints, active variables, reactions and daemons are used to uniformly represent system state and behavior. The use of an incremental constraint solver based on an enhancement of *DeltaBlue* makes it possible to run, at interactive speeds, complex applications composed of thousands of variables and constraints. The redraw time of the hardware is still the limiting factor on interaction speed.

We believe that *VB2* provides a good platform for prototyping and integrating a large variety of three-dimensional interaction metaphors to control all the different aspects of synthetic environments. We are currently extending the framework with time-varying constraints and tools for animation control in order to build a virtual reality animation system.

ACKNOWLEDGMENTS

We would like to thank Michel Gangnet, Geoff Wyvill and Russell Turner for reviewing this paper, and Angelo Mangili for his participation to the implementation and design of an early version of *VB2*.

BIBLIOGRAPHY

Balaguer JF, Mangili A (1992) Virtual Environments. In Thalmann D, Magnenat-Thalmann N (Editors) *New Trends in Animation and Visualization*, John Wiley and Sons: 91-105.

Borning A, Duisberg R, Freeman-Benson B, Kramer A, Woolf M (1987) Constraint Hierarchies, *Proc. OOPSLA*: 48-60.

Buxton WAS (1986) Chunking and Phrasing and the Design of Human-Computer Dialogues. In *Information Processing.* North Holland. Elsevier Science Publishers.

Buxton WAS (1990) A Three-state model of Graphical Input. In Diaper D, Gilmore D, Cockton G, Shackel B (Editors) *Human-Computer*

Interaction: Interact, Proceedings of the IFIP Third International Conference on Human-Computer Interaction, North-Holland, Oxford.

Conner DB, Snibbe SS, Herndon KP, Robbins DC, Zeleznik RC, Van Dam A (1992) Three-Dimensional Widgets. *SIGGRAPH Symposium on Interactive Graphics*: 183-188.

Fels SS, Hinton GE (1990) Building Adaptive Interfaces with Neural Networks: The Glove-Talk Pilot Study. In Diaper D, Gilmore D, Cockton G, Shackel B (Editors) *Human-Computer Interaction: Interact, Proceedings of the IFIP Third International Conference on Human-Computer Interaction*, North-Holland, Oxford: 683-687.

Freeman-Benson BM (1990) Kaleidoscope: Mixing Objects, Constraints, and Imperative Programming, Proc. ECOOP/OOPSLA: 77-87.

Freeman-Benson BM, Maloney A (1989) The DeltaBlue Algorithm: An Incremental Constraint Hierarchy Solver. In *Proceedings of the Eighth Annual IEEE International Phoenix Conference on Computers and Communications,* March.

Gobbetti E, Balaguer JF, Thalmann D, (1993) VB2: A Framework for Interaction in Synthetic Worlds. Submitted to *SIGGRAPH*.

Gobbetti E, Balaguer JF, Mangili A, Turner R (1993) Building an Interactive 3D Animation System. In Meyer B, Nerson JM (Editors) *Object-Oriented Applications*, Prentice-Hall.

Meyer B (1992) *Eiffel: The Language*. Prentice-Hall.

Myers BA (1989) User-Interface Tools: Introduction and Survey. *IEEE Software*. 6(1): 15-23.

NSF (1992), Research Directions in Virtual Environments, *NSF Invitational Workshop,* UNC at Chapel Hill, March 23-24: 154-177.

Rubine DH (1991), *The Automatic Recognition of Gestures,* PhD Thesis, CMU-CS-91-292, Carnegie Mellon University.

Sannella M, Borning A (1992) Multi-Garnet: Integrating Multi-way Constraints with Garnet, TR-92-07-01, Dept. of Computer Science, University of Washington.