AN OBJECT-ORIENTED METHODOLOGY USING DYNAMIC VARIABLES FOR ANIMATION AND SCIENTIFIC VISUALIZATION

Russell Turner, Enrico Gobbetti,
Francis Balaguer, Angelo Mangili,
Daniel Thalmann, Nadia Magnenat Thalmann

ABSTRACT

An object-oriented design is presented for building dynamic three-dimensional applications. This design takes the form of the Fifth Dimension Toolkit consisting of a set of interrelated classes whose instances may be connected together in a variety of ways to form different applications. Animation is obtained by connecting graphical objects to dynamic variables, which are able to change their values over time by responding to events. The Fifth Dimension Toolkit is the core of the Fifth Dimension Project, a research project for animating synthetic actors in their environment. The design philosophy and methodology of the toolkit are also described, as well as some of the implementation issues for the Silicon Graphics Iris 4D workstation.

Keywords: Object-Oriented, Animation, Scientific Visualization, Dynamic Variables.

1. INTRODUCTION

Complex dynamic three-dimensional graphics systems such as task-level animation systems and scientific visualization systems all involve various activities such as surface modeling, rendering, synchronization and motion control. A major problem in the development of such large systems is that they can become very difficult to maintain and extend.

A potential solution to this problem is to replace the traditional structured programming approach and top-down design strategy, with an object-oriented approach, which supports a bottom-up software design process. We use such an approach in the Fifth Dimension Project, a large research project in three-dimensional animation and visualization. The main objective of the project is the animation of synthetic actors in their environment, which involves a number of related areas of computer animation and scientific visualization. In particular, the following applications are being developed:

- animation of articulated bodies based on mechanical laws
- vision-based behavioral animation (Renault et al. 1990)
- hair rendering and animation
- intelligent object grasping
- facial animation
- personification in walking models (Boulic et al. 1990)
- synchronization in task-level animation
- deformation of flexible and elastic objects
- cloth animation with detection of collision

To coordinate efforts and allow good communication between the various applications, a toolkit of high-level dynamic graphical classes, both two and three dimensional, has been constructed. This toolkit, called the Fifth

Dimension Toolkit uses a uniformly object-oriented design for all its data structures, resulting in a high degree of integration between various applications.

In this paper, we discuss the goals of the Fifth Dimension Toolkit and describe its design. We also discuss the toolkit's design philosophy and its motivation, and briefly mention some of the implementation issues for the Silicon Graphics Iris 4D workstations.

2. GOALS OF THE FIFTH DIMENSION TOOLKIT

With the advent of workstations containing three-dimensional graphics engines, it is now easier to display relatively complex three-dimensional images in real-time. However, the accompanying software libraries usually operate at a very low level of drawing polygons. Although there are some commercial software systems available, they are often do not meet the needs of animators and scientific researchers who usually require highly specialized applications. As a result, a large part of their time is spent programming the basic user interfaces and three-dimensional data structures rather than their actual work. The Fifth Dimension Project involves several researchers working on related areas of computer animation and scientific visualization. To facilitate integration between various researchers' software and to allow the reuse of code, a core library of general-purpose, extensible software was required.

2.1 Previous Approaches

Many computer graphics research laboratories have an in-house "system" which includes such features as two and three dimensional modelers, renderers, image displayers and standard image file and model description formats. Some systems also incorporate various modeling and animation languages which can be compiled or interpreted by the system. Examples of such systems are described by Magnenat-Thalmann and Thalmann (1983), Chmilar and Wywill (1989), Hanrahan and Sturman (1985), Fiume et al. (1987) and Ostby (1989).

Although there are many advantages to using a high-level language, such as the ability to create procedural models, basing a system on a specialized language can make it more difficult to develop interactive tools. Also, it adds another level of translation to the system, requiring interpreters and parsers which have to be maintained and hindering extensiblity and reuse of code.

Another design approach is used by ConMan (Haeberli 1987, 1988) in which a number of small two-dimensional and three-dimensional application programs are connected together using inter-process communication. The various applications are "wired" together with the Connection Manager program in different ways to create different tools.

One of the difficulties with such an approach is the necessity of devising a proper inter-process-communication protocol so that all concepts and data structures can be communicated between the applications. A second problem is that, for shear reasons of efficiency, this method tends to force a coarse-grained modularity and the resulting large applications will still need to be built out of smaller software components.

The Pixar's animation system (Reeves et al., 1990) uses both these approaches. This system, which was used to create the animation *Tin Toy*, allows models to be created procedurally using a C-like interpreted language called ML. Interactive tool applications can be used to build models using the language. These tools communicate using a common database with shared memory and an interprocess communication scheme for passing messages between multiple tool processes.

2.2 Toolkit Approach

Another approach to building large graphics systems is the object-oriented toolkit. This is the basis of most current user-interface software packages such as Macintosh's MacApp, the X11 Window's Xt "widgets", Stanford University's InterViews, or NeXT's NextStep. These toolkits allow applications to create two-dimensional interactive "objects" on the screen with very complex behavior. Collections of these objects can be built up to implement user interfaces.

In some cases, user interface objects can be bound to one another so that manipulating one object will have an effect on the other. One of the first commercial object-oriented toolkits to incorporate dynamics into graphical objects was V.I. Corporation's DataViews (Kelly et al., 1988), which uses the concept of "graph" objects bound to dynamic variables to graphically display numerical data changing in real time. NASA's TAE Plus (Szczur, 1989) also uses the concept of "data-driven graphic objects" which display dynamic data on the screen. Both of these systems are essentially two dimensional.

2.3 Why an Object-Oriented Approach ?

All of these toolkits have been successful in part because they have used an object-oriented design philosophy. This is well-suited for the design of two-dimensional user-interface and dynamic graphics toolkits for several reasons. There is a close conceptual analogy between the dynamic object on the screen, which has spatial coherence and an innate behavior, and an instance of a class in memory, with its private variables and methods. Also, the self-contained nature and polymorphism of instances makes them ideal as modular, interconnectable components for building larger components or applications.

As pointed out by Micallef (1989), the kind of reusability which is given by the traditional library of subroutines is limited because the functionality provided by these pieces of software is fixed. A subroutine library can only be reused when exactly the same behavior as that provided by the existing subroutines is needed.

The concept of inheritance through class hierarchy, as described by Goldberg and Robson (1983) provides a mechanism for extensibility of classes and promotes reusability of code. Also, it provides an orderly way of handling special cases without affecting existing code. Finally, as shown by Meyer (1989), object-oriented programming by its nature tends to promote a bottom-up design approach rather than the traditional top-down one promoted by structured languages. This is certainly more appropriate for a research environment, where the final goals may evolve over time.

Object-oriented programming has already been successfully used in the field of computer animation. For example,"The ClockWorks" (Breen et al., 1987, 1989) animation system is implemented using an object-oriented methodology in C and uses message passing as a way to specify a scene's coreography.

All of these arguments strongly suggested to us that a large dynamic graphics system should be constructed using an object-oriented methodology to build a toolkit of classes, taking the object-oriented user-interface toolkit model and extending it to encompass dynamic three-dimensional objects. The toolkit should be developed in a bottom-up fashion, first building a set of reusable, general purpose classes and then going on to construct more powerful classes and complete applications.

2.4 Implementation Methodology

Our hardware consists of a network of Silicon Graphics IRIS 4D workstations. Ideally, the toolkit would be implemented in an object-oriented language, preferably an extension to C which would allow us to take existing C code and "encapsulate" it, turning it into a class. However, the basic concepts of object-oriented programming are not that complicated and it is usually not too difficult to build an object-oriented methodology on top of any standard computer language. So, for portability reasons, a methodology for doing object-oriented programming in C based on the methods described by Cox (1987) was developed.

This technique introduces a new typedef (`id`) which is a pointer to any object. Messages are implemented as functions in which the first parameter is the object that is the receiver of the message. Classes are implemented as external variables which are allocated at compile time.

For example, to send a new message to the Window class, creating a new instance of aWindow, the following statements would be used:

```
id aWindow;
aWindow = new_(Window);
```

By convention, message routines are distinguished with a terminating underscore and class object variables begin with an upper-case letter.

3. FIFTH DIMENSION TOOLKIT DESIGN

The design of the toolkit was based on several principles:

• **Device Independance.** The toolkit should be portable to other windowing systems and other machines with a graphics engine. This requires isolating the windowing system and device specific code to low-level routines and methods.

• **Extensibility.** The toolkit should not become a fixed programming interface, but should be able to be continuously extended through subclassing and the creation of new classes. This requires an open-ended design, with flexibility in the typing of objects.

• **2D and 3D integration.** There should be no conceptual distinction between the two-dimensional and three-dimensional graphical objects, or between the window system and the graphical models. This requires encapsulation of both two-dimensional window objects and three-dimensional models in a uniform way.

• **Inherently Dynamic Objects.** All graphical objects should have the potential to be dynamic in all their visible aspects and this dynamism should be built into the objects, not imposed from the outside. This requires classes specifically related to dynamic change over time, such as dynamic variables, and a mechanism for implementing the dynamics such as events.

3.1 Toolkit Classes and their Relations

The following diagram (Fig.1) shows the basic classes in a typical relationship. This type of diagram, which Rumbaugh (1987) calls an object-relation diagram represents classes as boxes and relationships between them as lines. The cardinality of the relationship is indicated by circles at the end of the lines. A filled circle represents a cardinality of zero to n, an unfilled circle represents a cardinality of zero or one, and no dot represents a cardinality of one. Subclass relationships are represented with triangles.

The classes on the right-hand side of the diagram mostly represent two-dimensional window system and user-interface objects while those on the left represent three-dimensional hierarchical objects, surface properties and lights. The Display class encapsulates the virtual workstation devices and it is the top of the graphical object hierarchy. The graphical link between the two portions of the diagram is the Camera, which represents the projection from the three-dimensional World to the two-dimensional Window.

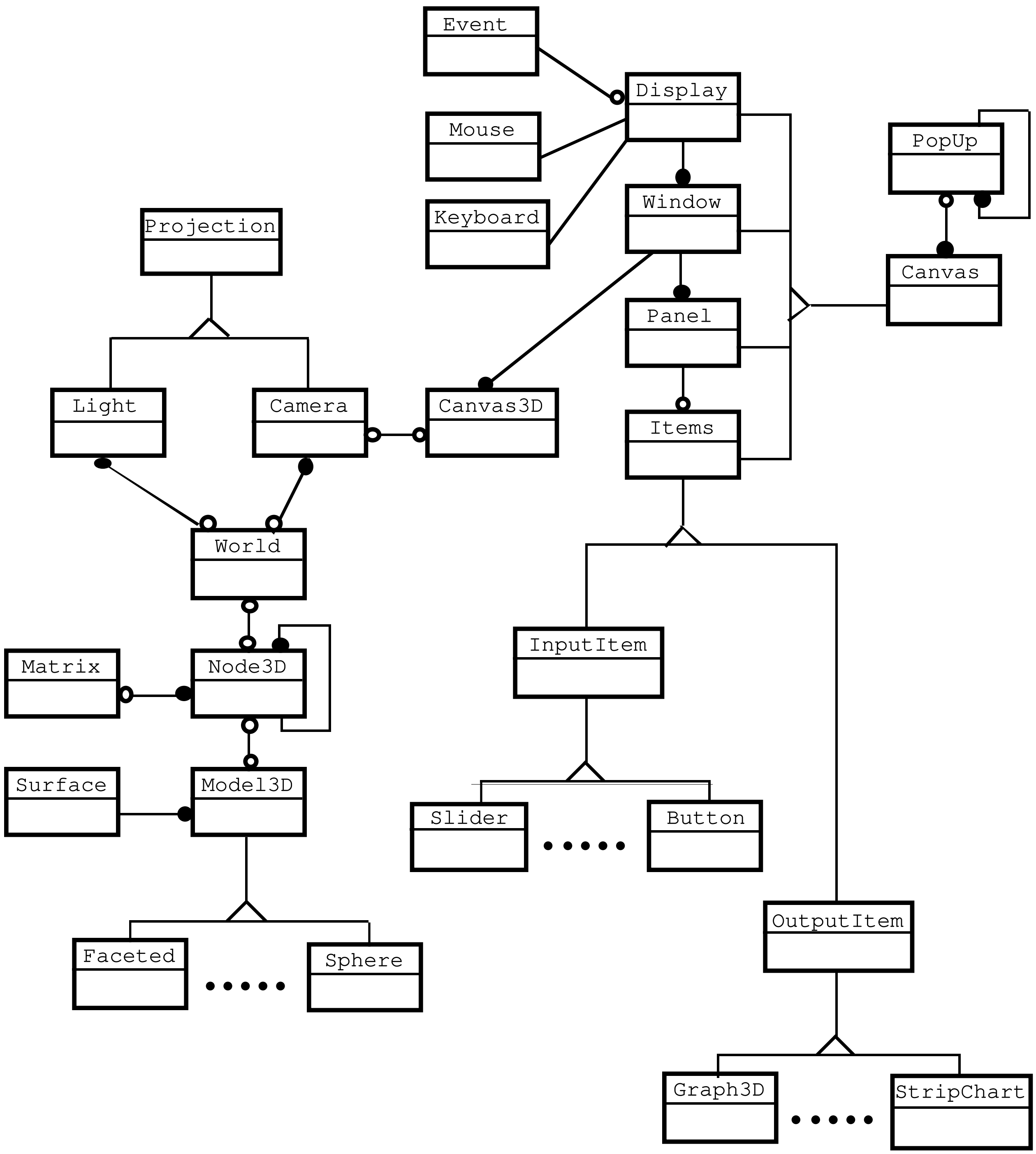

Fig.1 Object-relation diagram of the toolkit

All of the graphical class instances fit into a rough instance hierarchy going from the Display instance at the top down to the Model3D instances at the bottom. The Node3D instance can have indefinite levels of hierarchy representing three-dimensional hierarchical objects. One thing that all graphical classes have in common is the ability to respond to the `repaint_` message. This message cascades all the way down the instance hierarchy so that a `repaint_` message delivered to the Display instance will result in the whole application being redrawn.

Most of the two-dimensional classes represent standard windowing and user-interface objects found in many user-interface toolkits such as Xt or SunView: windows, panels, sliders, buttons, etc. These objects all fit into a hierarchy so that the display may own multiple windows, windows may own multiple panels, and panels can contain many input or output items. Since all of these classes inherit from the Canvas class, they all have similar behavior and design concepts. For example, they all have foreground and background color, position and dimensions on the screen, and all can be made invisible.

3.2 Event Messages

As in any object-oriented system, object instances in the Fifth Dimension Toolkit communicate with each other by sending messages. In order to respond to user-generated input and to implement the dynamic behavior of objects, we have defined a more formal type of message which we call an event message. This is a message sent by an object during the dynamic or interactive phase of a program's execution to indicate that it has changed state. All event messages have a single parameter which specifies the object that was the source of the event. To send an event message using the toolkit, the following syntax is used:

```
eventName_(destination, source);
```

where `eventName_` is the message selector, `destination` is the object to which the event message is sent, and `source` is the object sending the message.
For example, when a slider input object is moved by the user, it sends out a `newValue_` event message, giving the slider as source, to all objects requesting input from that slider.

This formalism for representing events, which is similar to the target/action mechanism in the NeXT machine's InterfaceBuilder (Webster, 1989), has several advantages. Since the destination object always knows the source of the event, it can query the source for whatever additional information it needs about the event. This separates the actual occurrence of the event from information passing about the event and shifts the information passing responsibility from the source object to the destination. Secondly, this strict form of event messages allows events to be represented as instances of the Event class, so that events can be manipulated as objects, and allows the distribution of events to be specified at run-time.

Objects that receive an event respond by calling the method associated with the event message's selector. In this way, all dynamic behavior of an object can be encoded within its event handling methods. It is therefore possible for the application program to have a minimal event handling loop as follows:

```
for (;;)
     {
     anEvent = returnNextEvent_(display);
     transmitEvent_(anEvent);
     update_(display);
     }
```

In this example, an event object is retrieved from the event queue, maintained by the Display object. The event is then transmitted to all objects that have expressed an interest in the event, which respond by calling their event methods. Finally, the graphical objects that changed their appearence as a result of the event are redrawn when the display is updated. In practical applications, the event loop is usually more complicated, with some control flow existing between the `returnNextEvent_` and `transmitEvent_` messages.

3.3 Dynamic Variables

A truly dynamic computer graphics system, as apposed to a static graphics system that is simply altered and then redrawn, must have a way for graphical objects to be intrinsicaly variable with respect to time. For example, if we have a robot consisting of a hierarchy of joints and we would these joints to move, we need to find a way for the joint matrices to change their value with respect to time automatically according to some external or internal source of data.

The concept of variables was already used in the field of computer graphics in the ASAS system (Reynolds, 1982), written in Lisp, where animated numbers are used, together with other facilities, to specify scripts for animation. The CINEMIRA animation language, described by Magnenat Thalmann and Thalmann (1982), generalized this idea by introducing the concept of "animated basic types". These are types of variables in the CINEMIRA language which can change their value as functions of time.

Hanrahan and Sturman (1985), introduced in their animation language the idea of parametric models in which a graphical object can be animated by binding its parameters to an animation system or directly to various user input devices. This concept was extended by Pixar (Ostby, 1989) in their Menv system by using “articulated variables”, which can contain other variables in a graph so as to form complicated expressions. A dependency list is maintained and is used to reevaluate only those values that have changed. These variables, like the parameters described by Hanrahan, are manipulated using a high-level interpreted language, called ML.

Rather than using a higher level language such as ML, the Fifth Dimension Toolkit takes the concept of “articulated variables” and implements them as classes. These classes, which we call dynamic variable classes, allow instances to be created that maintain a current mathematical value and can change their value over time. All dynamic variable instances are able to receive and transmit event messages, which change their values, and to respond to the `getValue_` message, which returns the current value of the instance.

Graphical objects can be made dynamic by attaching them to dynamic variables and dynamic variables can be interconnected themselves to form networks that calculate mathematical functions. Consistency is maintained by sending event messages and maintaining a graph of dependencies between the dynamic variables. In particular, every dynamic variable maintains a list of dependants, that is, other objects which depend on its value. Whenever a dynamic variable changes its value, it transmits a `newValue_` event message to all of its dependants. Each dependant object can respond to the event message as it sees fit. If it is another dynamic object, it can alter its own value accordingly and transmit its own `newValue_` message, or if it is a graphical object it can mark itself as out-of-date and redraw itself in response to the next `update_` message. In this way, multiple graphical objects can be bound to the same dynamic variable. For example, two sliders can be bound to ta single dynamic float variable so that when one slider moves, the other one moves together with it. Also, InputItems can be bound to OutputItems in various combinations.

Dynamic variables can also be bound to various types of data gathering classes so that their values will be changed in response to events outside the process. In this way, applications can respond dynamically to external data sources in real-time, or display event-driven simulations. It is also possible for dynamic variables to retain history of past values. This can be useful for dynamic objects that need to display some sort of historical process, for example, an OutputItem that emulates a strip-chart recorder.

Certain types of dynamic variables can act as functions. These maintain a list of dynamic variables which are its input parameters. When the function receives a `newValue_` event message from one of its parameters, it recalculates its own value, and sends out a `newValue_` event message to all of its dependants (see Fig. 2).

In general, graphical and dynamic objects can be assembled together into large networks, much like an analog control system or a collection of electronic components. In this way, complicated dynamics with many interrelated graphical objects can be driven by a few input variables.

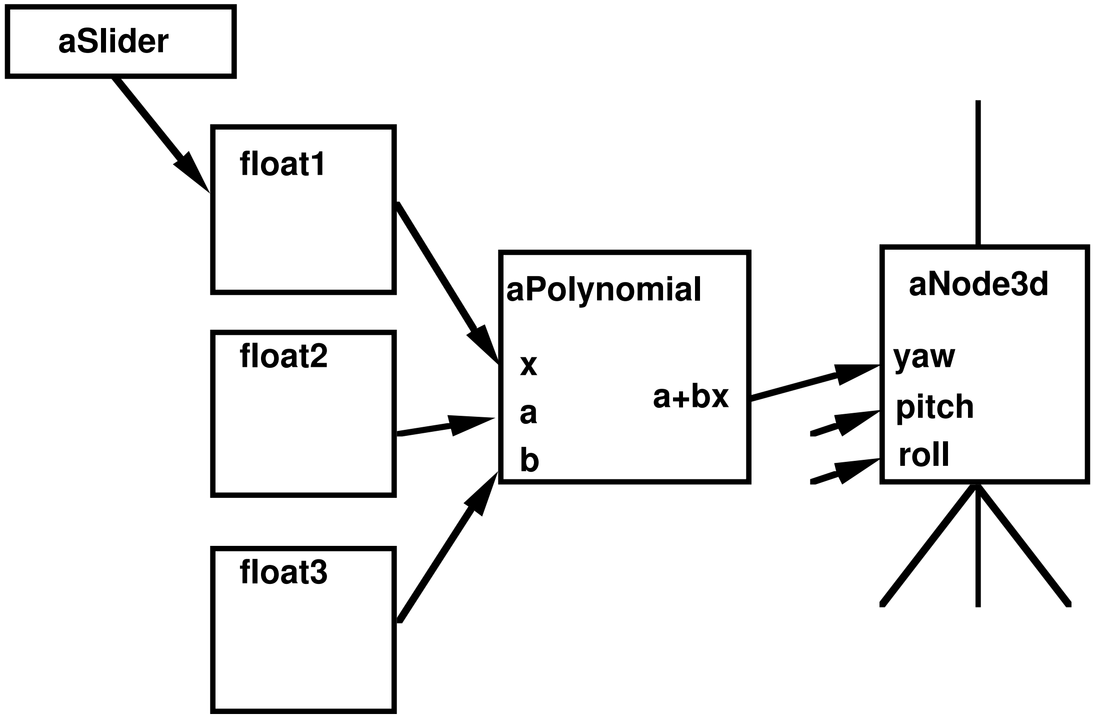

Fig 2. A network used to control the rotation of a three-dimensional graphic node by changing the value of a dynamic float with a slider.

4 Implementation

The Fifth Dimension Toolkit has been implemented on Silicon Graphics Iris 4D workstations. It was designed to support relatively portable applications so that when the toolkit is implemented on other windowing systems or machines with 3D graphics engines, the applications will compile and run properly. This is done by encapsulating all the device-specific graphics calls within a device-independent subroutine layer on top of which the toolkit classes are built. Special classes encapsulate several physical input devices such as Mouse, KeyBoard, SpaceBall and Digitizer.

As is shown in the diagram (Fig. 3), the underlying graphics engine is accessed by two very different types of software: a C graphics library for accessing the basic three-dimensional functionality of the graphics engine, and a NeWS/PostScript interpreter for implementing the user interaction and window system objects. We have used both software systems in our implementation: PostScript for implementing the two-dimensional InputItems using a set of classes inspired by the LiteItem toolkit (Densmore et al., 1987) and GL (the graphics library distributed by Silicon Graphics) for everything else. However, this division is not apparent to the application programmer using the Fifth Dimension Toolkit. The PostScript portion, for example, could be reimplemented using another language on top of other window systems such as X11 or even on top of the GL library.

One interesting consequence of this implementation is that the “look and feel” of the two-dimensional user-interface portion of the toolkit is implemented entirely in PostScript, and can be altered without having to recompile the C application. This is because PostScript is an interpreted language and all the definitions are downloaded at run-time.

Figure 4 shows an example of an application developed using the Fifth Dimension Toolkit.

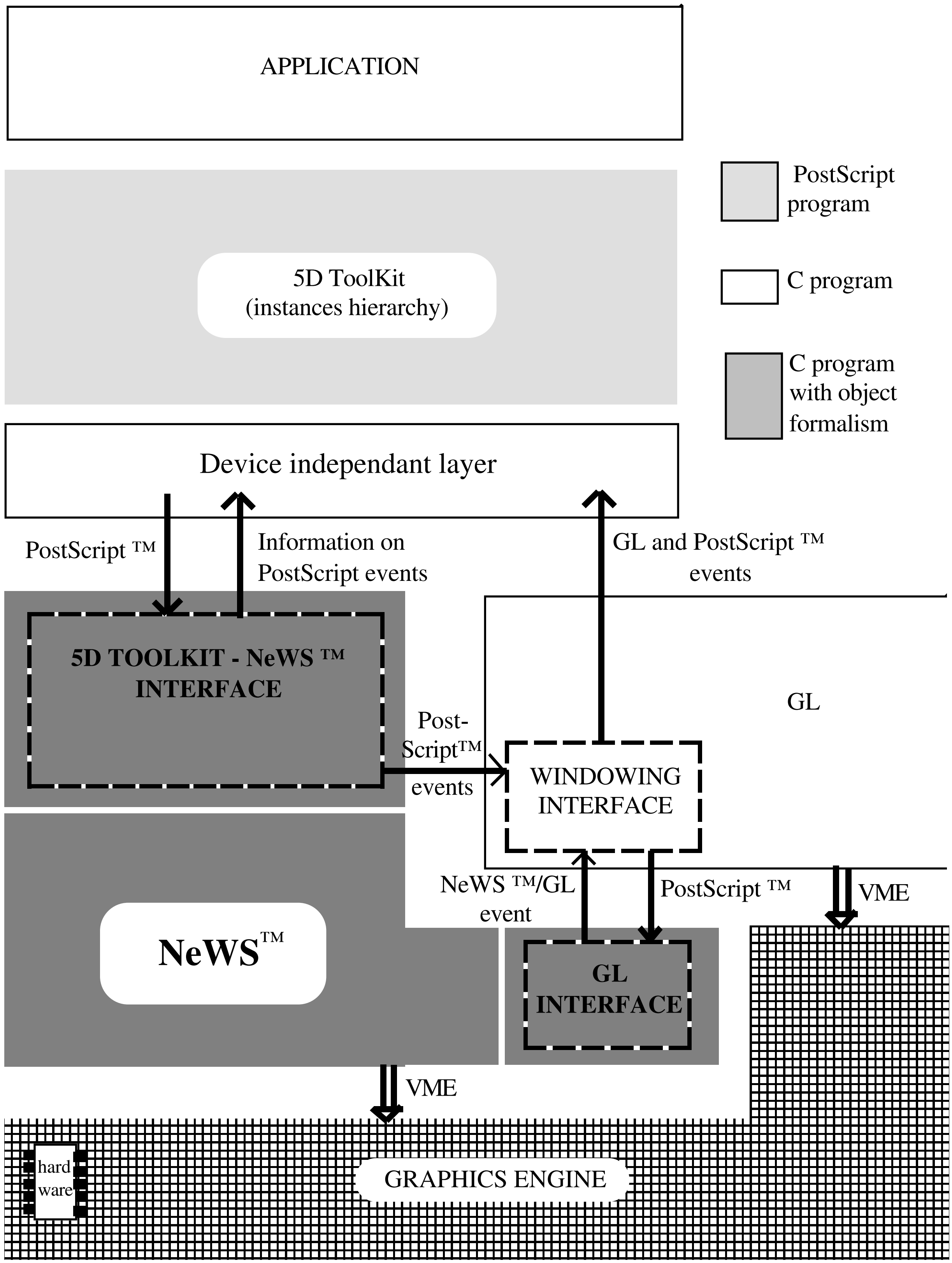

Fig.3 Fifth Dimension System Diagram

Fig.4 An example of application developed using the Fifth Dimension Toolkit

5. CONCLUSION

While programmers of standard two-dimensional applications have a large choice of user-interface toolkits, user-interface management systems and prototyping systems, the average scientific, engineering or animation software programmer writing software applications for super-computers or the latest “3D workstations” is required to use low-level graphics routines. Since many of the programmers writing three-dimensional dynamic applications are researchers and not professional computer programmers, their time spent coding up applications from scratch is time taken away from their research.

The Fifth Dimension Toolkit provides a working example of how an object-oriented toolkit concept of software building blocks can be applied to the construction of animation and scientific visualization software, resulting in substantial gains in software development productivity and code reuse.

ACKNOWLEDGEMENTS

We are grateful to Olivier Renault for his suggestions and help with the manuscript. The Fifth Dimension Project is partly supported by the Fonds National Suisse pour la Recherche Scientifique.

REFERENCES

Boulic R, Magnenat-Thalmann N, Thalmann D (1990) Human Free-Walking Model for a Real-time Interactive Design of Gaits, *Computer Animation '90*, Springer-Verlag, Tokyo

Breen D.E. et al. (1987) The Clockworks: An Object-Oriented Computer Animation System, *Proc. Eurographics '87*, North Holland, pp.275-282.

Breen D.E. Wozny MJ (1989) Message-Based Choreography for Computer Animation, *State-of-the-art in Computer Animation*, Springer Verlag, pp.69-82.

Chmilar M, Wywill B (1989) A Software Architecture for Integrated Modeling and Animation, *New Advances in Computer Graphics*, Springer Verlag, pp. 275-269.

Cox BJ (1987) *Object Oriented Programming: An Evolutionary Approach*, Addison Wesley.

Densmore OM, Rosenthal DSH (1987) A User-Interface Toolkit in Object-Oriented Postscript, *Computer Graphics Forum*, Vol.6, pp.171-180.

Fiume Eugene et al.: "A Temporal Scripting Language for Object-Oriented Animation". Proc. Eurographics '87, North Holland, pp. 283-294.

Golberg A, Robson S (1983) *Smalltalk-80: The Language and its Implementation*, Addison Wesley.

Haeberli PE (1987) A Data-Flow Manager for Interactive Graphics, *Iris Universe*, fall pp. 3-5..

Haeberli PE (1988) ConMan: A Visual Programming Language for Interactive Graphics, *Proc. SIGGRAPH '88, Computer Graphics*, Vol.22, No4, pp. 103-111.

Hanrahan P, Sturman D (1985) Interactive Animation of Parametric Models, *The Visual Computer,* Vol. 1, No4, pp. 260-266.

Kelly M, Aczel TG, Turner R, Dee D (1988) *DV-Tools User's Guide*, version 6.0, V.I. Corporation, Amherts MA

Magnenat-Thalmann N, Thalmann D (1983) The Use of High-Level 3-D Graphical Types in the Mira Animation System, *IEEE Computer Graphics and Applications* 3(9), pp. 9-16.

Magnenat-Thalmann N, Thalmann D (1983b) Actor and Camera Data Types in Computer Animation, *Proc. Graphics Interface '83*, pp. 203-209.

Meyer B (1989): From Structured Programming to Object-Oriented Design: The Road To Eiffel, *Structured Programming*, Vol. 10, No1, pp.19-39.

Micallef J (1988): Encapsulation, Reusability and Extensibility in Object-Oriented Programming Languages. *Journal of Object-Oriented Programming* Vol. 1 No. 1, pp. 12-35.

Ostby EF(1989) Simplified Control of Complex Animation, *State-of-the-art in Computer Animation*, Springer Verlag, Tokyo, pp. 59-67.

Reeves WT, Ostby EF, Leffler SJ (1990) The Menv Modelling and Animation Environment, *Visualization and Computer Animation Journal*, John Wiley, Vol,1, No1 (July 1990)

Renault O, Magnenat-Thalmann N, Thalmann D (1990) A Vision-Based Approach to Behavioural Animation, *Visualization and Computer Animation Journal*, John Wiley, Vol,1, No1 (July 1990)

Reynolds CW (1982) Computer Animation with Scripts and Actors, *Proc. SIGGRAPH'82, Computer Graphics* Vol.16, No3, pp. 289-296.

Rumbaugh JL (1987): Relations as Semantic Constructs in an Object-Oriented Language, *Proc. OOPSLA '87*, pp. 466-481.

Szczur MR (1989) TAE Plus: Transportable Applications Environment Plus, *Xhibition89*, San Jose, California, June 1989.

Webster BF (1989) The NeXT Book. Addison Wesley.

The Authors

Russell Turner is a researcher at the Computer Graphics Laboratory of the Swiss Federal Institute of Technology in Lausanne, Switzerland. He received his B.S. in Physics and his M.S. in Computer and Information Science from the University of Massachusetts at Amherst. He has also worked as a software engineer for V.I. Corporation of Amherst, Massachusetts. His research interests include computer animation, user-interfaces and object-oriented programming. He is a member of IEEE and ACM.
E-mail: `turner@elma.epfl.CH`

Enrico Gobbetti is a researcher at the Computer Graphics Laboratory of the Swiss Federal Institute of Technology in Lausanne, Switzerland. He received his diplôme d'ingénieur informaticien from the same institute. His research interests include visualization, computer animation and object-oriented programming.
E-mail: `gobbetti@elma.epfl.CH`

Francis Balaguer is a researcher at the Computer Graphics Laboratory of the Swiss Federal Institute of Technology in Lausanne, Switzerland. He received his diplôme d'ingénieur informaticien from the Institut National des Sciences Appliquées (INSA) in Lyon, France. His research interests include computer animation, user-interfaces and object-oriented programming.
E-mail: `balaguer@ligsg2.epfl.CH`

Angelo Mangili is a researcher at the Computer Graphics Laboratory of the Swiss Federal Institute of Technology in Lausanne, Switzerland. He received his diplôme d'ingénieur informaticien from the same institute. His research interests include computer animation and software engineering.
E-mail: `mangili@elma.epfl.CH`

The authors can be reached at the following address:

Computer Graphics Laboratory
EPFL-LIG
CH-1015 Lausanne, Switzerland