XP: An Authoring System for Immersive Art Exhibitions

Dave Pape, Tomoko Imai, Josephine Anstey, Maria Roussou, Tom DeFanti
Electronic Visualization Laboratory (EVL)
University of Illinois at Chicago
851 S. Morgan St., Room 1120
Chicago IL 60607-7053
pape@evl.uic.edu, timai@evl.uic.edu, anstey@evl.uic.edu, mr@fhw.gr, tom@uic.edu

Abstract. In this paper we describe a software system for building interactive virtual environments, particularly ones for virtual reality art works. It is meant to allow teams composed of experienced programmers and non-programming designers to work together on projects. It is an object-oriented framework, built upon existing toolkits for VR, real-time graphics, and audio. A number of common application features and tools are provided, simplifying world creation.

1. Introduction

Many artists are interested in creating virtual reality art pieces but are daunted by the complex computer coding that is required. Typically, artists work with engineers, but this scenario does not give an artist the opportunity to experiment and work in a hands on fashion with the application – also an engineer's time can be expensive! To alleviate this problem, we designed XP, a framework for virtual environment applications.

The XP system grew from software developed for the "Multi-MegaBook in the CAVE" application [1], and was further refined during the development of "Mitologies" [2]. These two applications are both large-scale environments. They involve hundreds of megabytes of models, texture-maps, and audio clips; the environments cover a large virtual space, and include multiple scenes. Most of the artists involved were experienced with tools such as Softimage, Photoshop, or basic Unix, but were not expert computer graphics programmers. Our goal was to create a framework which included many of the features common to virtual art environments, one that would allow experienced VR programmers to build tools needed for the features unique to a specific application, and allow artists to create the final environment by assembling the appropriate pieces.

2. Background

Our target applications are ones that run in the CAVE virtual reality system [3], or the ImmersaDesk [4]. The default input devices for these systems are a wand, with 3 buttons and a small joystick, and a 6 degree-of-freedom tracker, which tracks the user's head and the wand. These inputs define the forms of interaction available to a user in the CAVE; an application can react to the position of the user or the wand, or to button presses or joystick use. Specific interactions which have become common in applications include navigation

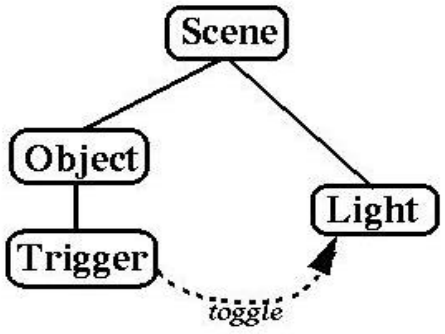
through the environment, picking up and dropping objects, transitioning between scenes when the user moves through a portal, and "clicking on" objects to trigger special reactions.

The code for typical CAVE applications uses Silicon Graphics' IRIS Performer [5], the CAVE libraries, the VSS sound library [6], and C++. Performer is a real-time 3D visual simulation toolkit; one of its most important contributions for the design of XP is its scene graph. The scene graph is a hierarchical representation of an application's visual database; it is a tree or directed acyclic graph, where the leaf nodes contain data such as geometry and light sources, and the internal nodes provide special features such as grouping, transformations, and selection. The CAVE libraries provide transparent access to the virtual reality hardware, taking care of trackers and input devices, and handling the details of correct stereoscopic rendering for whatever display device is being used. VSS is a client/server system for interactive playback and control of sounds. We based our new system on these tools, defining standard, simplified interfaces to their major features.

3. Design

The system is divided into two major aspects – the text file(s) defining an application as a collection of nodes and their attributes, and the lower-level C++ classes which implement the nodes. With this division, it is possible to split the work of world-creation between experienced programmers and non-programmers. The programmers create any application-specific nodes in the underlying class collection, while other team members build the full application by plugging together nodes in the text files. It also makes it easier to re-use code between applications, because the code is all in modular XP nodes with standardized interfaces.

Application authors create a virtual environment in XP by editing a scene text file. The text file is a high-level description of the environment's scene graph. It lays out the structure of the scene graph, listing the nodes and their hierarchical relationships, assigns attributes to the nodes, and defines interactions among the nodes. Figure 1 shows a simple example file, and the corresponding scene graph. User interactions, and behaviors involving multiple nodes, are defined by events and messages. In XP, an event is considered to be something that happens "within" a node – that is, something might happen which a particular node type is interested in; a node of that type will check during each frame whether this event has happened, and will signal when it occurs. Messages can then be sent from the node to other nodes in response to the event. For instance, a trigger node could detect when a wand button-press event has occurred, and send the message "toggle" to a light, to turn the light on or off in response to the user's action. When an event/message combination is defined in the scene file, the message can be given a delay; the message will then be sent the given




```
group ()
  {
  object (file=LightSwitch.iv)
    {
    wandTrigger (eventMessage=
          "button1,Lamp,toggle")
    }
  light (name=Lamp)
  }
```

Figure 1. A simple scene graph with message

number of seconds after the event occurs, rather than instantaneously. This feature is useful when complex responses to events are desired, where several things happen in succession.

When a node is defined in the text file, options for it are listed in parentheses after the node's type name. The options generally define attributes which will vary between different instances of a node, such as the color of a light or the X/Y/Z translation of an object.

An XP scene graph is built of nodes which consist of a Performer node plus behavior. All node classes are derived from xpNode, which defines a common class interface for the nodes. The basic xpNode interface includes methods for performing per-frame updates to the node, receiving messages from other nodes, parsing the node's description in a text file, and resetting and disabling the node. Node classes extend these methods as needed, to define the behavior specific to that class.

The standard classes that are part of the basic system include *transform*, *switch*, *object*, *light*, and *sound*; there are also a set of *trigger* classes which respond to user actions. *transform* nodes are used to translate, rotate, and scale the parts of the world which are under them in the scene graph. By default, a *transform* is static, but subclasses of *transform* are often defined to create dynamic transformations, for application-specific movements of objects (playing back keyframed animations, for instance). A *switch* node is used to turn parts of the virtual world on or off at run-time, such as in a transition between scenes.

object nodes encapsulate 3D object models, which can be in any modeling format supported by Performer. Performer provides database-loaders for a number of common formats, and new custom-built ones can be easily added; because many of the artists we work with use the Softimage animation package, we created a loader for Softimage's hrc format, which was not part of the standard Performer loaders. *object* nodes have a number of options which may be enabled or disabled; these include being grabbable (i.e. a user can pick up and drop the object), being "walls" for collision detection, or being "floors" for terrain-following navigation. They can also be marked as un-drawn, to create invisible walls or floors specifically to control the user's movement. *light* nodes are point or infinite light sources which illuminate the scene; they can be turned on or off dynamically. *sound* nodes contain audio clips which can be played in response to messages; being a part of the scene graph, they have a 3 dimensional position, and their amplitude can be varied based on the user's distance from the sound source. Many systems, such as VRML [7], define sounds as point sources, whose amplitude decays in a spherical or ellipsoid pattern around the point. In XP, we have extended this model to allow sounds to occupy volumes (spheres or boxes), within which the sound's amplitude is constant; outside the volume, the amplitude decays normally. This is not intended to be realistic, but proves useful in many applications for such things as having a sound emitted uniformly by a large object, or creating a background sound which fills an entire room.

The *trigger* classes include nodes which detect when the user enters or exits a space, when the wand enters or exits a space, when a button is pressed while the wand is within a space, or when the user points at something. These actions will generate corresponding events, which can be used in the scene definition file to send messages to other nodes, to respond to the user's actions.

When an XP application is running, the scene graph is traversed during the course of each frame. This traversal visits each node in the world, calling its update method to perform any computations and changes for continuous behaviors. If a node detects that an event has occurred, any messages associated with this event are placed in a queue. Once the scene traversal finishes, the queue is checked; any messages which should take place during the current frame (those that should be sent immediately, or those whose delay time has expired) will be sent to their recipient nodes.

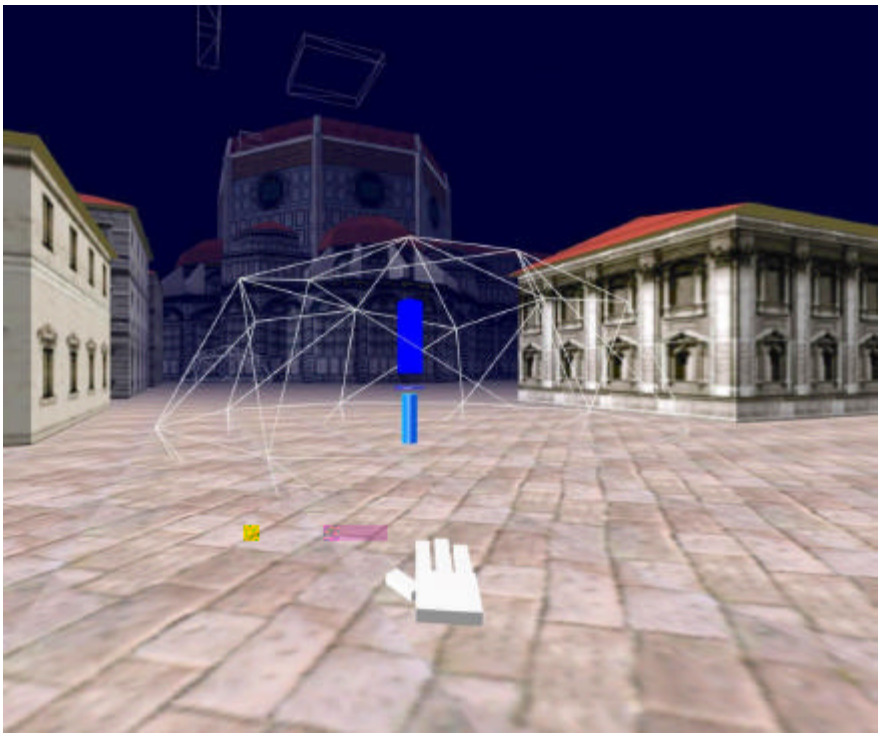


Figure 2. Multi-MegaBook in debug mode

All nodes include a method to reset them to their initial state. This feature is important for running applications in public shows. Applications which have a specific narrative or flow need to be re-initialized whenever a new user, or group of users, enters the environment. Because of the scale of the applications we have been working on, we cannot simply exit the program and reload everything from scratch – long delays are not acceptable at showtime. Also, inexperienced users can sometimes lose their way in an application, or can find unanticipated “holes” in the environment; often in these cases, a quick reset of the world is the best way to rescue them.

Many nodes also have a debugging state, which is used during development and testing. When debugging is enabled, additional elements are drawn, showing normally invisible aspects of the scene. For example, triggers will draw their bounding volumes, so that the developer can check their size and placement in the scene; their state changes are indicated by changing colors. Events and significant messages are printed to the terminal, so that the flow of the application can be monitored. Figure 2 shows a view of the Multi-MegaBook environment in debugging mode.

In addition to the scene graph defining the virtual world, other major elements of the XP framework include the navigator and world nodes, which are automatically created parts of the scene. The navigator aids the user’s movement through the world. Although the user can move around physically, within the tracked region of the CAVE, exploring a large scale environment requires moving over much greater distances. This is accomplished by effectively transporting the CAVE through the virtual world, under the control of the navigator. The navigator provides tools for both user-controlled and application-controlled navigation. Typical user-controlled movement is directed with the wand – the user points the wand in the direction to move, and uses the joystick to control the speed of movement, or to turn left or right. Application-controlled navigation features include teleporting to a specific location, following a spline path, or attaching the CAVE to a node in the scene graph. The navigator also provides optional collision-detection, to prevent users from passing through walls, and terrain-following, to keep users walking on the ground of the

Figure 3. A virtual tour-guide

environment. Further application-specific navigation features can be added by sub-classing the standard navigator node class.

The world node serves as the root of the scene graph, and provides an interface to some global attributes. It can be used to change the background sky color, enable or disable fog,and vary the clipping plane distances. It also encapsulates all of the scene text file parsing code, and controls the scene graph traversals.

4. Applications

The XP system was used to implement the “Multi-MegaBook” and “Mitologies” applications, which have been shown to large audiences at such shows as SIGGRAPH, Ars Electronica, and ISEA. Several other applications since then have been or are being created with the system. We describe here some of the interesting aspects of the implementation of these works.

In very large virtual worlds, several of the design considerations involve either controlling or aiding the user in navigating through the environment. Although theoretically users could move anywhere in a virtual environment, in some cases the artist creating a specific world will want to limit them. “Mitologies” has a general flow to its experience, requiring a specific introduction and conclusion, where the user’s navigation is completely controlled. At the beginning, it is attached to a boat which carries users down a river; at the end, users are “dropped” out of the last room and carried back to the beginning point; in between, however, the navigation is unconstrained. One of the central scenes to the “Multi-MegaBook” is an extensive city; users are free to explore the city as they wish, but we don’t want them to get lost or miss the important parts. To help them, keyframe-animated characters are placed in the city to act as virtual tour-guides; when users approach the guide, it will lead them to a nearby point of interest (Figure 3).

We make extensive use of switches and triggers to control how much of an overall world is active. The triggers detect when the user is in a certain region, and only activate those parts of the environment that are nearby. Although culling and level-of-detail are often

useful in limiting how much is drawn in a purely visual database, we must also consider the computational load of nodes' behaviors; using switches allows us to handle this.

The basic XP system is designed for single-user applications. We have, however, begun extending it further to support networked, multi-user worlds. Using the CAVERNsoft networking architecture [8], a scene graph is replicated among multiple CAVEs; changes made by one user are automatically shared with all others. This approach was used in building the V-Mail (virtual mail) system for collaborative design [9]. New node classes were created which can record user actions and spoken comments, and then send them to remote collaborators, who may be in the shared environment at different times. In the future, we plan to include such networking of the scene graph at the core of the system.

5. Conclusion

The XP system provides a framework for creating large scale, interactive virtual reality applications. By dividing the development of applications into two distinct components – the coding of nodes that encapsulate specific behaviors, and the assembling of these nodes into a scene – XP allows teams comprising artists and programmers to work on projects efficiently. It has been used to develop several successful artistic virtual worlds. Although it was originally developed for art applications, we believe the general system will be useful in building a wide range of virtual worlds.

Acknowledgments

The virtual reality research, collaborations, and outreach programs at EVL are made possible through major funding from the National Science Foundation, the Defense Advanced Research Projects Agency, and the US Department of Energy; specifically NSF awards CDA-9303433, CDA-9512272, NCR-9712283, CDA-9720351, and the NSF ASC Partnerships for Advanced Computational Infrastructure program.
CAVE and ImmersaDesk are trademarks of the Board of Trustees of the University of Illinois.

References

[1] F. Fischnaller and Y. Singh. Multi Mega Book. In catalog of Ars Electronica Festival 97, Linz, Austria, September 1997.
[2] M. Roussos and H. Bizri. Mitologies: Medieval Labyrinth Narratives in Virtual Reality. In the proceedings of 1st International Conference on Virtual Worlds, Paris, France, July 1-3, 1998.
[3] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. In Proceedings of SIGGRAPH '93 Computer Graphics Conference, ACM SIGGRAPH, August 1993, pp. 135-142.
[4] M. Czernuszenko, D. Pape, D. J. Sandin, T. A. DeFanti, G. L. Dawe, M. Brown. The ImmersaDesk and Infinity Wall Projection-Based Virtual Reality Displays. Computer Graphics, Vol. 31 No. 2, May 1997, pp. 46-49.
[5] J. Rohlf and J. Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. In Proceedings of SIGGRAPH '94 Computer Graphics Conference, ACM SIGGRAPH, August 1994, pp. 381-395.
[6] S. Das, T. A. DeFanti, and D. J. Sandin. An Organization for High-Level Interactive Control of Sound. In Proceedings of the International Conference on Auditory Display '94, Santa Fe, February 1994.
[7] The Virtual Reality Modeling Language. International Standard ISO/IEC 14772-1:1997.
[8] J. Leigh, A. Johnson, T. DeFanti. CAVERN: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments. Virtual Reality: Research, Development and Applications, Vol 2.2, December 1996, pp 217-237.
[9] T. Imai. The Virtual Mail System. MS Thesis, Electrical Engineering and Computer Science Department, University of Illinois at Chicago, August 1998.