Tools for Collaborative VR Application Development

Adrian Haffegee[1], Ronan Jamieson[1], Christoph Anthes[2], and Vassil Alexandrov[1]

[1] Centre for Advanced Computing and Emerging Technologies, The University of Reading, Reading, RG6 6AY, United Kingdom
sir04amh@reading.ac.uk
[2] GUP Linz, Johannes Kepler University Linz, Altenbergerstraße 69, A-4040 Linz, Austria

Abstract. This paper introduces a tool set consisting of open source libraries that are being developed to facilitate the quick and easy implementation of collaborative VR applications. It describes functionality that can be used for generating and displaying a Virtual Environment (VE) on varied VR platforms. This is enhanced to provide collaboration support through additional modules such as networking. Two existing VR applications which make use of these tools are described. Both were developed effortlessly over a short period of time, and demonstrate the power of these tools for implementing a diverse range of applications.

1 Introduction

With the growing range and power of VR systems, increasing numbers of users are gaining access to VEs. These are virtual spaces created within the system, that a user can enter and then interact with. Linking these using various technologies expand the virtual domain, and bestow the users with quasi person to person interaction.

A potentially limiting factor in the development of collaborative VR applications is the time, effort and expense of their implementation. A new tool set consisting of the CAVE Scene Manager (CSM), networking utilities and additional component libraries have been designed and implemented. These aid developers in creating their applications by providing the key functionality of generating, displaying and enhancing a VE.

Work relating to this development will be discussed in Section 2. This will be followed by Sections 3 and 4 respectively; first with a description of the CSM, and then the additional tools. Two sample applications currently using this tool set are then mentioned in Section 5, before the paper concludes in Section 6.

2 Related Work

Designing tools for collaborative VR application development requires the consideration of several key areas, most of which have existing implementations.

Networked VR viewers, such as Coanim which comes with the CAVERNsoft toolkit [1], allow the display of geometries for different remote users in a basic Networked Virtual Environment (NVE). In this type of application users can modify attributes in the shared environment (e.g. transformations for geometries) or trigger animation events. Further descriptions of this particular type of VE can be found in [2, 3].

Scenegraph libraries (OpenGL Performer [4], OpenSG [5]), provide a structured database of entities and their relationships in the scene. Typically these entities are rendered graphically enabling representation of potentially complex environments.

Navigation and interaction within the VE is required to allow users to interface with the environment. A good overview of different navigation methods can be found in [6].

Avatars, which are used to graphically represent remote users, both in their position and actions, are a highly important feature for collaborative environments. A description of how these can be implemented is found in [7].

Many different types of VR hardware such as CAVEs [8] or head mounted displays can be used for the display of VEs. A generic application should allow operation with many of these differing systems, providing hardware independence. Offerings providing similar functionality include those from CAVELib [9], and VRJuggler [10].

3 The CAVE Scene Manager (CSM)

The CSM handles the VE's scenegraph representation (described below), as well as any display's video rendering. This module is built on top of a scenegraph library, forming an integral part of the tool set.

Although scenegraph independence would ideally be desired for the tool set, OpenSG [5] was chosen for the initial design and implementation due to its cluster support, structure and for being open source.

3.1 Configuration

Despite its name, the CSM can be used for displaying on a wide range of VR systems and not just CAVEs. Many parameters of these systems, for example sizes, graphics configuration, or tracking dimensions, change with each installation. To allow for these differences the CSM uses a configuration file which describes the particular setup. With this approach, by just referencing a different configuration file the same application can be run on any VR platform.

To allow simple integration with existing VR installations, the configuration file format was made compatible with that used for CAVELib configuration. While the CSM does not utilize all the CAVELib parameters, it uses those sufficient for generating multi-wall stereo images, ignoring those it does not require.

In addition to the standard CAVELib configuration options, additional optional parameters have been included for advanced CSM configuration.

3.2 Internal Scenegraph Representation

The internal structure of the CSM contains a node tree for the positioning of the user within the VE. Figure 1 shows the OpenSG representation of these nodes within the CSM. The circles represent the nodes which are used to describe the graph hierarchy, while the rectangles are node cores determining the type of node functionality.

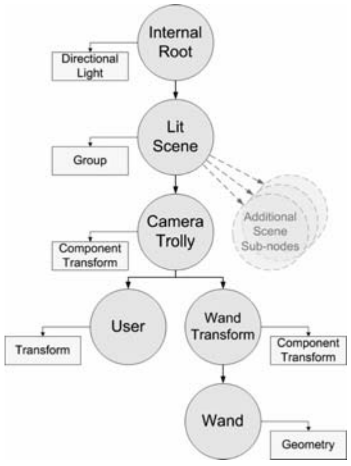

Fig. 1. Node tree for the CAVE Scene Manager

The *internal root* node has a directional light source core for a headlight, with its beacon being the *lit scene*. This ensures every node below the internal root will be lit if the headlight is on.

An OpenSG component transform is used to position a *camera trolley* node. This represents the navigated position in the scene of a trolley upon which the cameras are mounted. Two child nodes from their camera trolley parent are used to show the positions of the two tracking devices; the user position from head tracking, and also the wand position as generated by the CAVE. Finally a basic geometry node is added to display the default wand within the environment. The user, wand and camera trolley can be moved independently, and the CSM has functions for updating these positions as required. A common technique in the CAVE is to move the camera trolley using the joystick on the controller, with user and wand positions coming from the tracking devices. For desktop operation all these movements would come from a keyboard/mouse combination.

A VE application passes the root node of the its scene to the CSM, where it gets added as a child node of the internal root. This is useful for setting a scene that has a fixed location and does not move, such as a street or a room. However, often additional entities are needed to represent objects or remote users within the scene. Functionality has been provided to add and remove the node tree representations of these, and also to change their base position and orientation as required. These functions reference the entity through its root node without the CSM needing any knowledge of the entities' composition. This allows complex animated geometries to be easily incorporated into the scene.

In a similar way the default wand geometry can be overridden to display any node tree while still tracking any movement of the wand. This could be used to model a particular type of pointer, a sophisticated moving hand, or even a hand using or manipulating some additional object.

3.3 Display Walls

The CSM's main task is to display images, optionally stereo, on one or more screens (monitors or projection walls). Virtual cameras are placed in the scene, and are used to determine the images being rendered.

The CSM configuration determines how many cameras exist in the scene. It has fields for display wall configuration which describe the position of the viewing device relative to the user's position. Defaults have been provided for the regular CAVE walls which form a cube around the user. When these are not sufficient, such as for a curved screen, the system is configured with the corner coordinates of the screen or projection plane. One camera is used for each wall being rendered, facing in the appropriate direction for that wall.

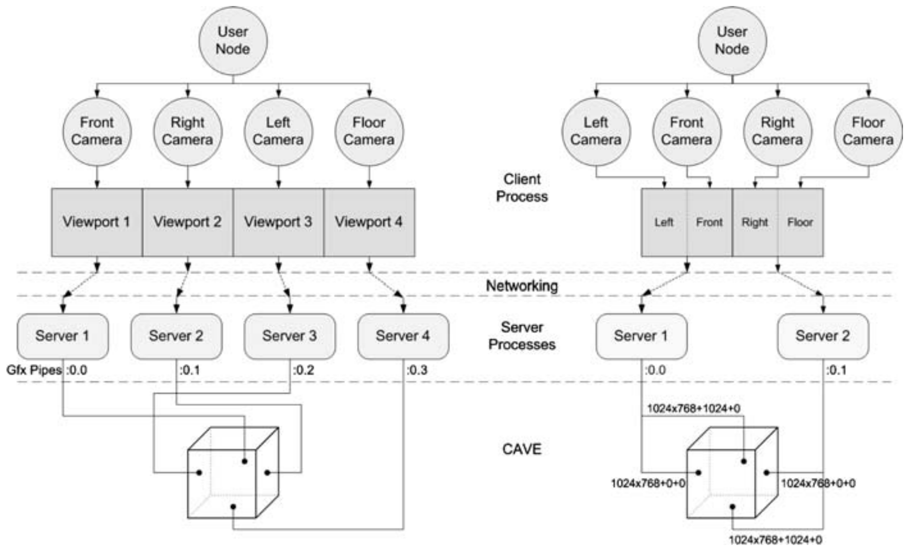

Fig. 2. Client/Server Cluster Window for a 4 pipe (left) and 2 pipe (right) CAVE

3.4 Client/Server Clustering Windows

The OpenGL Utility Toolkit (GLUT), which is used for displaying the graphics windows can only output to one graphics pipe per process. This is an issue when multiple displays are required. An alternative to a multiple process scheme uses clusters, with the full environment being constructed through a network of individual computers, each responsible for its own portion.

Both of these approaches are supported in the CSM through OpenSG's client/server clustering. A client process handles the scenegraph manipulations, and one or more server processes render the graphics. These processes can exist on the same system, operate remotely, or a combination of the two. This is achieved through CSM configuration.

When clustering, the views generated by the client are split over a grid with as many cells as there are servers. Each server only requires the data for its cell, which it renders for its connected display.

Figure 2 shows two possible approaches for clustering for a four wall CAVE. In the architecture on the left each wall is driven through its own pipe. For this the client window is split into 4 viewports, with each camera placing its output into one of these. Each server receives a quarter of the client window, therefore receiving the data from the camera for the wall it represents.

The architecture on the right is slightly different in that the system only has two pipes, with each of these driving two walls. In this case the client window is split in half, with each half being sent to a different server. Two viewports are combined onto each of these halves to display the correct image split over two walls. In this example the split is vertically down the middle of each graphic pipe.

4 Additional Functionality

In addition to the scene and display functionality provided by the CSM, several other useful tools are also included. These include a command parser for accepting user input, which would then be used to activate certain commands or functionality. Individual application configuration files can also be utilized, and work along the same lines as for CSM configuration.

Three of the more important tools are avatars, networking and navigation/interaction, which we describe below.

Avatars. Two types of avatar currently exist. Simple avatars have been included with limited realism, however they do make use of range based level of detail, allowing a potentially large number of them to be simultaneously active with little processing overhead. Alternatively, highly graphical avatars with realistic motion can also be used. These increase observer immersion as they move and perform pre-set actions in highly realistic manner.

Networking. A networking infrastructure has been developed providing low latency, and reliability. This has been integrated into a topology based on the descriptions from [11], making it possible for remote clients to reliably communicate. Internally, the networking tools control default network handling, including the necessary messaging for remote avatar representation. To this, functionality has been provided to allow sharing of user defined messages. As will be seen in the sample applications these messages can be used for advanced collaboration.

Network simulators and logging/debug routines have also been developed for assistance in application development.

Navigation and Interaction. Each time the displayed frame is updated the application polls the tracking information from shared memory. While the head and wand tracking data are passed to the CSM, the controller information is used for navigation and interaction. Current navigation makes use of the joystick and wand orientation to move within the scene. It is anticipated that this could be further extended through the integration of a navigation framework such as that described in [12].

Basic collision detection between the user and the scene has also been provided for when one of the wands buttons had been activated. Either as a result of this, or directly from the interaction with the wand it is possible to transmit events to any remote users, thereby forming the basis of user interaction and collaboration beyond that achieved from a simple visual response.

5 Applications

5.1 Flooding Visualisation

This application demonstrates the benefits gained through visualization in an immersive environment. As part of the EU CROSSGRID project, it uses the GRID for landscape flooding simulation [13]. The CSM provides an integral part of this application, allowing easy viewing on the various types of VR hardware.

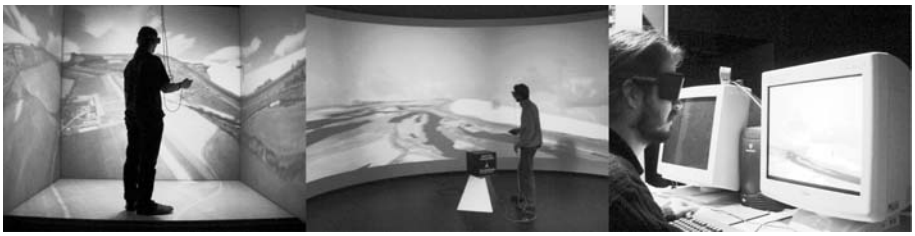

Fig. 3. Interaction in the CAVE, Curved Screen and on a desktop system

The data to be simulated is calculated with Globus middleware on the GRID, and then transferred to the display clients through GRIDFTP. The simulation makes use of two types of VRML 2.0 models. The first makes up the landscape and consists of a height map geometry of the terrain, upon which a photographic image has been superimposed. This provides a realistic environment with good spatial orientation.

The second group of models, are used for representing the flood water. A separate set of triangle meshes are provided for each time stamp during the flooding process, and these are applied in sequence to the terrain to simulate the flood.

For efficiency in maintaining a realistic frame rate (>30fps), all geometry data is preloaded into the scene on start-up. The scene root has two children; one for the terrain, and the second a switch node which parents each individual flooding time slice and controls which image is being rendered. The switch node steps through these slices in order, providing smooth animation. On a desktop system the simulation is controlled via a keyboard, whereas fully immersive systems will also use their wand. A flying navigation mode is used, allowing the scene to be viewed from any angle. The keyboard or wand is also used for animating forward or backwards at a pre-defined speed, or for pausing the simulation. Figure 3 demonstrates the simulation being used on three different VR platforms.

5.2 Virtual House Application

The second application has been developed as an easily extendible VE that allows remote users to interact collaboratively with objects within a virtual house environment. It is built on top of the VR tool set which manage display and navigation, both in the CAVE and on the desktop. It uses the networking functionality to distribute each user's motion and event data, and the avatars to display these locally.

Current functionality shares tasks associated with opening and closing a door in the virtual house. Demonstration of this can be seen in Figure 4, which show a CAVE user about to manipulate a moving door that has been opened by a desktop user. In each case the remote user position is depicted by an avatar. Virtual objects that can be interacted with in this environment are defined as

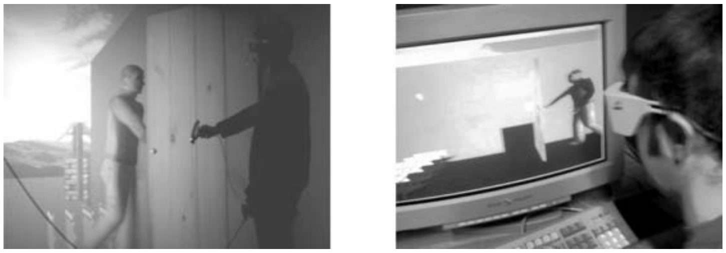

Fig. 4. Collaborative door opening in the Virtual House application

Dynamic Interaction Objects (DIO). They contain attributes and functionality relative to how they can perform. For instance, the door DIO has attributes declaring its axis of rotation and the degree of rotation allowed. For ease of management and flexibility these attributes are contained in a text file which is read during execution.

On initialization the scenegraph is traversed, and DIOs are created for those defined from the text file. Simple collision detection is used to determine if a user is interacting with one of these DIOs, and if so the relevant attributes are modified. Additionally, an Object Manager component continuously services the

DIOs, allowing them to modify the VE as required. Events, such as the door starting to open, are sent over the network for remote notification.

Future improvements will increase functionality of the DIO, allowing it to respond to multiple input events, thereby developing realistic multi-user concurrent collaboration. Further study is also required of potential latency issues, both in terms of avatar movement and event passing.

6 Conclusion

This work describes a useful set of tools that have successfully been used to implement different types of collaborative VR environments. However it should be noted that these form the basis of a potentially larger and more advanced offering. Additional work could see advanced networking, the integration of a navigation library, improved avatars and further collaboration support, all which would allow equally simple, but more advanced application development.

References

1. Park, K., Cho, Y., Krishnaprasad, N., Scharver, C., Lewis, M., Leigh, J., Johnson, A.: CAVERNsoft G2: A toolkit for high performance tele-immersive collaboration. In: VRST, Seoul, Korea, ACM Press (2000) 8–15
2. Macedonia, M.R., Zyda, M.J.: A taxonomy for networked virtual environments. IEEE MultiMedia **4** (1997) 48–56
3. Matijasevic, M.: A review of networked multi-user virtual environments. Technical report tr97-8-1, Center for Advanced Computer Studies, Virtual Reality and Multimedia Laboratory, University of Southwestern Lousiana, USA (1997)
4. Rohlf, J., Helman, J.: IRIS Performer: A high performance multiprocessing toolkit for real-time 3D graphics. In: SIGGRAPH, ACM Press (1994) 381–394
5. Reiners, D.: OpenSG: A scene graph system for flexible and efficient realtime rendering for virtual and augmented reality applications. PhD thesis, Technische Universität Darmstadt (2002)
6. Bowman, D.A., Koller, D., Hodges, L.F.: Travel in immersive virtual environments: An evaluation of viewpoint motion control techniques. In: VRAIS (1997) 45–52
7. Badler, N.I., Phillips, C.B., Webber, B.L.: Simulating Humans: Computer Graphics Animation and Control. Oxford University Press, New York, NY, USA (1992)
8. Cruz-Neira, C., Sandin, D.J., Defanti, T.A., Kenyon, R.V., Hart, J.C.: The CAVE: Audio Visual Experience automatic virtual environment. Communications of the ACM **35** (1992) 64–72
9. VRCO website. http://www.vrco.com/ (2004)
10. Bierbaum, A.D.: VRJuggler: A virtual platform for virtual reality application development. Master's thesis, Iowa State University, Ames, Iowa (2000)
11. Anthes, C., Heinzlreiter, P., Volkert, J.: An adaptive network architecture for close-coupled collaboration in distributed virtual environments. In: VRCAI, Singapore (2004) 382–385

12. Anthes, C., Heinzlreiter, P., Kurka, G., Volkert, J.: Navigation models for a flexible, multi-mode VR navigation framework. In: VRCAI, Singapore (2004) 476–479
13. Hluchy, L., Habala, O., Tran, V.D., Simo, B., Astalos, J., Dobrucky, M.: Infrastructure for grid-based virtual organizations. In: Proceedings of the International Conference on Computational Science, Part III, LNCS 3038, Springer Verlag (2004) 124–131