DEVELOPMENT OF *RUTOPIA 2* VR ARTWORK USING NEW YGDRASIL FEATURES

Daria Tsoupikova and Alex Hill
Electronic Visualization Laboratory, University of Illinois at Chicago, Chicago, IL, USA

Keywords: Virtual Reality (VR), Ygdrasil, VR art, CAVE®, tele-immersion.

Abstract: Ygdrasil is a programming framework used by artists and computer scientists worldwide to create networked multi-user virtual reality (VR) worlds and tele-immersive art projects. Recently added advanced rendering modules and scripting language improvements extend the user's creative control. This paper describes the development of the VR art project Rutopia 2 using these new features.

1 INTRODUCTION

Multi-user, networked virtual reality technology enables real-time collaborative science and engineering. It also serves as a unique art medium for the enrichment of the human spirit. Through the use of high-speed networks, powerful graphics workstations and tracking technology, multiple users can coexist in an interactive alternate reality of sight and sound. The development of collaborative VR environments requires numerous technologies including 3D modelling software, computer graphics (CG) rendering techniques, high-speed networking, sound spatialization and voice conferencing. Artists working in this medium need development tools that abstract the process to a level that allows them to concentrate on their artistic expression instead of the technology supporting it.

The Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago (UIC) developed the world's first projection-based virtual reality room, called the CAVE®, in 1992. In the mid-nineties, EVL began to connect CAVEs together over high-speed networks to create real-time tele-collaborative virtual environments. These initial development efforts relied heavily on the expertise of computer scientists working in collaboration with artists. To streamline the process for non-expert users, a script-based framework for shared world development called Ygdrasil was developed in 1997 by EVL's Dave Pape (Pape, 2002).

The Rutopia 2 project explores the aesthetics of virtual art and traditional Russian folk art of wood sculpture and toys, and the decorative painting styles of the Russian regional art centers of Palekh, Khokhloma, and Dymkovo. These art centers are famous for superb workmanship, the diverse methods and techniques, distinctive features and use of narrative imagery—mostly fairytale. Each distinct style is identifiable by its ornamental pattern, colour palette and the choice of materials. Rutopia 2 aesthetics is based on the generalized outlines, decorative colour schemes, and flamboyant colours inspired by these styles (Ovsiannikov, 1967).

Recent improvements to the Ygdrasil framework now allow artists to create advanced virtual worlds with minimal technical assistance. The framework initially required specialized C++ knowledge to develop special purpose modules for all but the most basic projects. This paper describes recent scripting language improvements and high-end rendering technique modules that helped to create the enriched dynamics and content of the VR art project Rutopia 2.

2 YGDRASIL FRAMEWORK

The Ygdrasil framework combines a transparent shared scene graph with a script-based interface and dynamically linked modules. Ygdrasil utilizes a parallel scene graph along side the OpenGL Performer scene graph and updates distributed nodes using the QUANTA networking library developed at EVL. Ygdrasil generated objects can encapsulate complex behaviours performed on Performer scene

Tsoupikova D. and Hill A. (2007).
DEVELOPMENT OF RUTOPIA 2 VR ARTWORK USING NEW YGDRASIL FEATURES.
In *Proceedings of the Second International Conference on Computer Graphics Theory and Applications - AS/IE*, pages 225-228
DOI: 10.5220/0002081502250228

graph objects such as switches, transforms, and geometric objects. Ygdrasil scripts allow the creation and initialization of scene objects through the passing of messages. Additional messages can be passed between objects at runtime to dynamically change the content. Dynamic messages are initiated using events generated by the individual nodes.

For example, a *userTrigger* node might generate the event "enter" when a user enters a region of the environment. Any number of messages can be assigned to the "enter" event and subsequently sent to any other node in the scene graph. Events can also encapsulate additional information relevant to the event itself. For example, in the following script segment

```
userTrigger trigger1(volume(sphere),
   when(enter,
        $user.teleport(100 0 0)))
```

the variable *$user* is replaced with the name of the user node that entered the trigger. Ygdrasil nodes related to external devices such as a Wanda(TM) or keyboard input also generate events useful for programming interactive behaviours.

2.1 Scripting Language Improvements

In the original implementation of Ygdrasil, the information available when responding to events was limited to a small set of data passed along with the event. It soon became apparent that access to the internal state variables of each node would allow greater flexibility in programming interactions. Unfortunately, the original Ygdrasil system architecture did not facilitate accessing internal state variables because there was a mismatch between the stored internal state and the messages sent to the node.

The internal state of each Ygdrasil node was comprised of any number of database keys, each requiring a unique identifier within the QUANTA networking middleware. The available key datatypes include strings, floats, integers, booleans, and several different sized float arrays. This internal state was originally optimized to minimize the amount of data required for updating networked client nodes. For example, several messages indicating the state of collision detection for a geometry node (i.e. *wall(true)*, *floor(false)*) were OR-ed into a single short integer for transfer to client nodes. Because messages are often aggregates of several data types, this mismatch required logic to parse the incoming message into internal database variables and further logic to retranslate state variables back into the message format for save-to-file or runtime access within events. In order to unify the relationship between message and state, the message architecture was reorganized around a system of aggregate data types. The data now transferred to client nodes directly represents the aggregate data type of the messages that adjust them. By registering messages and their corresponding data types with the system in this fashion, a simple heuristic can easily recreate message arguments. In the following script segment

```
userTrigger trigger1(volume(sphere),
   when(enter,trans1.event(teleport,
                        user=$user)))

transform trans1(position(10 20 0),
   when(teleport,
        $user.teleport($position0
                       $position1
                       $position2)))
```

the variable *user* is passed to the transform node and used in combination with the position state variables to teleport the user to the current position of the transform.

In order to facilitate the calculation of intermediate variables within the scripting language, an arrayed float *value* node was created and sub-classed for various operations. Using recursive descent on binary and unary trees allows calculations to be constructed within the existing scene graph data structure. In the following script segment

```
add(when(changed,
        trans1.position(0 $value 0)))
   {
   multiply()
         {
         value height1()
         value(set(2.0))
         }
   value(set(10.0))
   }
```

a value of 10.0 is added to the product of *height1* and 2.0. The final result generates a "changed" event and the state variable can be used to generate a message to another node. In addition to floating point operations, a set of *and*, *or*, *not* and *boolean* test nodes (i.e. *lessThan*, *greaterThan*) support a full range of boolean logic operations. These math nodes do not affect the resulting visualization and, therefore, their location within the scene graph is arbitrary.

2.2 Advanced Rendering Techniques

Even with access to state variables and intermediate variable calculations, artists often find that they need advanced rendering techniques to realize their artistic vision. Techniques that fall into this category are vertex morphing, real-time texture manipulation, alternate rendering viewpoints, clipping planes and stencil buffer operations. These techniques must be componentized carefully in order that they remain flexible and useful within the scripting environment. We use two strategies to address these needs; segmentation of functionality and separation of rendering object from rendering source.

The majority of early dynamic nodes for Ygdrasil encapsulated the time based dynamics within the node itself. For instance, the *spinner* node only accepted a message to adjust the period of a full rotation but could not be paused or reversed. By segmenting the time manipulation into a general-purpose *timer* node and passing only an orientation value to the spinner we can gain better control of the dynamics. Nodes for path following, vertex morphing, material properties and others can now have their dynamics paused, reversed, and looped easily with this new implementation. Moreover, segmenting the manipulation of large data arrays helps to retain the power of techniques without sacrificing flexibility. Programs such as Photoshop and Maya are useful for manipulating large arrays of pixels and vertices respectively. Our vertex-morphing node only takes a keyframe position and morphs between the vertex positions defined within two 3D model files. And, our texture application node applies a secondary texture to an arbitrary location on an existing texture for producing effects such as burns, bullet holes, or x-ray vision through surfaces. Users do not manage the values of individual pixels, they merely apply a smaller image onto another image at a specified X and Y location. In both cases, the manipulation of individual pixels remains in the realm of more special purpose programs while the expressive power of the rendering technique can easily be manipulated dynamically within Ygdrasil.

Many rendering techniques rely both on an object located within the scene and operation on some subset of the geometry in the scene. The clipping plane node, for instance, must both position the clipping plane in the scene and indicate the subset of the scene graph that is subject to clipping. Our *viewTexture* node renders a subset of the scene from an alternate viewpoint and applies it to a texture object within the scene. And, our *stencilBuffer* node must specify both a graphical object used to create the stencil mask and a subset of the scene graph to be rendered subject to the mask. In order to accommodate these dual needs we locate the rendering node at one location within the scene and give it the name of a node indicating the subset of the scene graph it should apply to. In the following script segment

```
stencilBuffer(node(stencilGroup))      {
  object tree(file(tree.pfb))
}
group stencilGroup(){
  object moon(file(moon.pfb))
}
object terrain(file(terrain.pfb))
```

the *tree* object defines the geometry shape of the stencil mask, the *moon* object is rendered into the resulting mask, and the *terrain* is rendered normally. As a result, the user can see the moon only within an area defined by the rendering of the tree.

3 RUTOPIA 2

Rutopia 2 is a virtual reality art project describing a magic garden with interactive sculptural trees that create portals to distant worlds. It was conceived as a virtual environment linked to a matrix of several other unique virtual environments that together create a shared network community. The goal of the interaction scheme is to avoid the preliminary instructions usually required to familiarize the user with the virtual environment and its rules of exploration. User interaction is based on the participant proximity to interactive locations while the wand interface is used only to control the direction of movement. The project implementation utilized Ygdrasil, OpenGL Performer 3.2, CAVElib and the Bergen spatialized sound server on an Intel Linux PC running SUSE 10.0 and connected to an Ascension Flock of Birds tracker.

Figure 1: The Island world with the trees.

A series of 3D modular sculptural trees, each consisting of dozens of rectangular screens, appear in the main environment and serve as portals to the other linked environments (Fig. 1). Users can "grow" three trees in the monochrome island world by moving within the proximity of each tree. Each tree appears as a rapid sequence of flipping and rotating rectangular screens expanding out and upward. Once all the trees are fully grown, their screens turn into windows and the island changes from monochrome to colour. Each window shows the view of the remote environment connected to it. Just as we can look through a window and see the outside, the user can look through each of the screens to see a house world consisting of imagery found in traditional Russian fairytales and folk art. By moving his or her head completely through one of the virtual screens, the user enters the connected environment (Fig. 2).

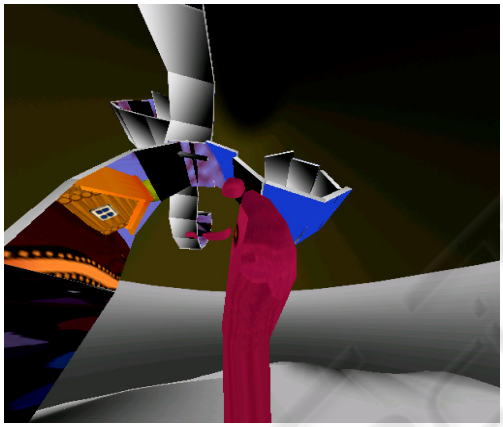

Figure 2: An avatar peeks into the details of the remote world.

3.1 Utilizing Rendering Nodes

The Rutopia 2 project was realized using the latest improvements to Ygdrasil including state variable access and logical operations. The windows of the trees were made using the new *stencilBuffer* node. This node acts as a mask covering the areas outside the windows so that only the selected window area allows a view to the other world. The other world consists of two objects, the rendered object and the stencil object. The rendered object is the geometry of the remote place the user can see through the window. The stencil object forms the viewing window and is utilized by the stencil buffer mask so that the user can see only a portion of the rendered object through the region defined by the stencil object. Each third window-hole on a tree is connected to the same view of the house world in an alternating fashion. Participants can recognize and visually connect lower and upper parts of the remote house world projected on the different level windows to appreciate an even broader view of the remote environment.

4 CONCLUSION

The development of VR environments is interdisciplinary in nature and requires both artistic and scientific skills. For the last several years, artists and scientists have used the Ygdrasil programming framework to create networked multi-user virtual worlds and art projects. Recent scripting language improvements and advanced rendering techniques within the Ygdrasil framework empower the creative freedom of artists in order that they may realize their creative visions. These recent improvements to Ygdrasil greatly contributed to the development of dynamic interactions and advanced rendering effects in the Rutopia 2 project.

ACKNOWLEDGEMENTS

We would like to acknowledge the support of the University of Illinois at Chicago's Electronic Visualization Laboratory (EVL), Geophysical Center Russian Academy of Sciences (GC RAS), Global Ring Network for Advanced Applications Development (GLORIAD) and San Diego State University. This material is based in part upon EVL work supported by the National Science Foundation (NSF), notably equipment awards CNS-0224306 and CNS-0420477 and international networking infrastructure awards OCI-0229642 and OCI-0441094. The CAVE, CAVElib and Wanda are registered trademarks of the Board of Trustees of the University of Illinois.
http://www.evl.uic.edu/yg/
http://www.evl.uic.edu/animagina/rutopia/rutopia2/

REFERENCES

Pape D., Anstey J., Dolinsky M., Dambik E., 2002. Ygdrasil--a framework for compositing shared virtual worlds. In *Future Generation Computer Systems, Special Issue IGRID 2002*, pp. 1041-1049.

Chinese puzzle consisting of geometric shapes to be reassembled into different figures.

Ovsiannikov I., 1967. Russian folk arts and crafts. Moscow, Progress Publishers.