The Flatland Architecture, An Open Source Visualization/Virtual Reality Development Environment

Kathleen H. Kihmm, kihmm@hawaii.edu[1], Kenneth L. Summers, summers@hpc.unm.edu[2], Andrei Sherstyuk, andrei@hawaii.rr.com[1], Timothy Eyring, teyring@comcast.net[2], Steven Smith, sas@lanl.gov[3], Paul M. Weber, pmw@hpc.unm.edu[3], Thomas Preston Caudell, tpc@ece.unm.edu[2,4]

1. Telehealth Research Institute, University of Hawaii, 2. Center for High Performance Computing, The University of New Mexico, 3. Los Alamos National Laboratory, 4. Department of Electrical and Computer Engineering, The University of New Mexico

I. Introduction

Flatland is an open source visualization/virtual reality application development environment developed collaboratively by the University of New Mexico, Los Alamos National Laboratory, and University of Hawaii. Flatland allows software authors to construct and users to interact with arbitrarily complex graphical, aural and haptic representations of data. Flatland is multi-threaded, allowing the system to take advantage of computer systems with multiprocessors and shared memory. The main thread can spawn children at runtime to support the available perceptual modalities. For example, a graphics (visual) thread, a sound (hearing) thread and/or a haptics (touch) thread. Optional tracker threads may be launched to allow the user to use 3D interaction devices such as head or eyepoint motion tracking.

Flatland is written in C/C++ and uses the standard OpenGL graphics language extensions and standard GLUT library for window, mouse, and keyboard management. A custom external sound server provides sound, and a MIDI interface has been integrated to add dynamic sound synthesis methods and functionality. Flatland utilizes an internal data structure consisting of its own object type; objects are embedded in graphs that maintain and facilitate the integrity of the system. Flatland uses dynamically linked shared libraries to assemble modules that construct or modify the virtual environment (VE) for specific applications.

II. Flatland Architecture

A Flatland application consists of the Flatland core and an arbitrary number of modules. The core consists of a set of routines and services that provide basic functionality for control and maintenance of a geometrical transformation graph. This graph is composed of independent objects, each of which provides its own graphics, sound, and haptics to the system. The Flatland core coordinates these services for all objects and maintains the geometrical relationships stored in the graph.

A. Graphs

The transformation graph is part of a higher-level structure referred to in Flatland as a universe. The universe contains the transformation graph, a flat database of objects in the graph, and a reference to the graph vertex (object) that is currently acting as the root of a hierarchically organized tree. The tree is an instance of the graph with one vertex promoted into the role of root. The root vertex's coordinate system is used as a "virtual camera lens" location for graphical image, aural, and haptic rendering. Flatland maintains the geometric transforms between objects, and the graph is reconfigured on the fly into different instantiations.

In order to reduce lag between a user action and consequential rendering, multiple graph representations can be instantiated in the universe. For example, if an object has been moved in a particular scene, other objects may be affected or collided by this move. The brute force method to detect a collision would be to reorient the graph so that the object being moved becomes the root. This transforms all other objects into the moved object's frame of reference. If these transformed objects lie along the Z-axis of the moved object, then collisions have occurred. The next operation would be to reorient the graph again with the eye object as the root node. The above process can be parallelized, minimizing the interaction time. In this case, two parallel graphs are used, one for the drawing from the eye object, and one for collision detection from the moving object.

B. Modules

A module in the context of Flatland is a relatively self-contained collection of objects, functions, and data that can be dynamically loaded (and unloaded) into the system and attached to the graph. The module's initialization function is executed at load-time, and is responsible for creating and attaching the module's objects to the graph.

C. Controls

Once a module is loaded, users can interact with objects in several different ways. Currently there are four interfaces for the control of objects: 1) GLUT pop up menus in the main viewer window, 2) the console keyboard, 3) Flatland 2D control panels, and 4) external systems or simulations.

D. Auto Configuration

Flatland is made aware of loadable modules through the use of a configuration file. This file contains the list of modules, their initial placement in the graph, and their user specifiable parameters. Module libraries contain a standard interface function that may create zero or more Flatland objects as well as launches user-defined threads.

III. Conclusion

The flexibility of the Flatland environment offers developers a highly configurable tool for quick prototyping and development of VE applications. Flatland runs on all UNIX systems that support POSIX threads and have OpenGL and GLUT development support. Linux and IRIX operating systems are currently being used as development platforms.

IV. Acknowledgements

The authors would like to acknowledge the historical contributions of Chris Woody and Paul Hubbard, and the support of the University of New Mexico Center for High Performance Computing, Department of Electrical and Computer Engineering, The Boeing Company, Los Alamos National Laboratory, and Sandia National Laboratory.