HECTOR – Scripting-Based VR System Design

Gordon Wetzstein[1,3] Moritz Göllner[2,3] Stephan Beck[3] Felix Weiszig[3] Sebastian Derkau[3] Jan P. Springer[3] Bernd Fröhlich[3]

[1] University of British Columbia [2] Ceetron GmbH [3] Bauhaus-Universität Weimar

Abstract

Modern VR systems embrace the idea of scripting interfaces for rapid prototyping of applications. HECTOR goes one step further: the entire core of the VR system is written in PYTHON, easily gluing interchangeable high-performance C++ libraries, a module-based system infrastructure and a scripting-based application layer. Thus, the time consuming and error prone compile-debug cycle for application and system development becomes obsolete — both, the infrastructure and the application layer, can be extended or modified even at run-time.

Keywords: Virtual Reality, Scripting, Distributed Graphics

Introduction

Recently, a variety of VR systems [Blach et al. 1998; Tramberend 1999; Bierbaum et al. 2001] have been proposed that allow applications to be rapid-prototyped via script interfaces. Due to their interpreted nature, scripting languages are developer friendly, easy to debug, and applications are modifiable during execution. While most of the existing VR systems provide scripting interfaces, their internal structure often follows a monolithic design principle, which makes changing of individual system parts, such as the renderer or physics engine, problematic.

VR Juggler [Bierbaum et al. 2001] introduced a modular design that allows the system to be extended quite easily. Hector follows a new design paradigm based on the same idea: while time-critical system components are still implemented in C++, a scripting-based glue layer provides a module loading and unloading mechanism, distributed event-driven communication within the system as well as an application development interface. Thus, novel design concepts for VR systems can be prototyped in a light-weight scripting environment during run-time.

Concepts

Individual HECTOR modules are threads that implement functionality for rendering, 3D sound synthesis, event distribution, physical simulation and input device management. Most of these modules utilize existing libraries such as OpenGL|Performer [Rohlf and Helman], OpenSceneGraph (http://www.openscenegraph.org/), Open Dynamics Engine (ODE, http://www.ode.org/), or Spread (http://www.spread.org/). A specific module implementation can be chosen in a configuration script depending on availability and operating system. When the system is started only the micro kernel and the communication module are initialized. All other modules are loaded upon request by newly created objects in the virtual environments. All objects have module-dependency lists that trigger reference counters in the kernel which also unloads unnecessary modules.

All communication within the system is event driven. Whenever an object is created, destroyed or its attributes are modified events are distributed to all components that have been registered in the communication module for this event type. This allows a natural extension for distributed applications where local events are spread to all HECTOR stations in the network and handled as if they were created locally. PYTHON's pickle interface provides a powerful way of event serialization where even internal references to PYTHON objects are automatically kept consistent within remote applications.

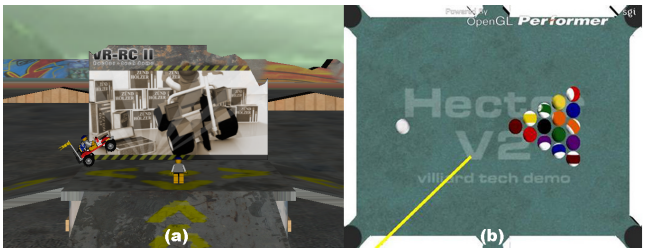

Figure 1: Two Hector applications: a physically-based buggy racer (a) and a virtual pool game (b).

HECTOR's distribution mechanism is implemented on IP multicasting. New applications broadcast a join request to all existing stations and are then updated with the current state of the virtual environment through reliable TCP connections. In order to guarantee a consistent environment in realtime all event changes are multicast with unreliable UDP messages. Furthermore, the network module keeps a hash table that associates each distributed object attribute to its most recent change event. This ensures that the attributes can be internally updated independently of how fast they can be send and that no deprecated information floods the network. For distributed physically-based applications generally only one station runs the physics engine and redistributes the results of the simulation to other stations, even though individual objects are connected to input devices on those.

Results and Discussion

HECTOR is a lightweight, open source, scripting-based, distributed, and platform independent VR system. It is written in PYTHON and glues several high-performance C++ libraries. Extensions to the framework can be prototyped rapidly during runtime. HECTOR is freely available at www.sourceforge.net/projects/hector. The system has been successfully tested with various applications, such as a distributed physically-based buggy racer (cf. figure 1) or a virtual pool game that is displayed on a TFT display horizontally mounted on a table and controlled by a tracked real pool queue.

References

BIERBAUM, A., JUST, C., HARTLING, P., MEINERT, K., BAKER, A., AND CRUZ-NEIRA, C. 2001. VR Juggler: A Virtual Platform for Virtual Reality Application Development. In *Proceedings of IEEE Virtual Reality 2001 Conference*, IEEE Computer Society, 89–97.

BLACH, R., LANDAUER, J., RÖSCH, A., AND SIMON, A. 1998. A Highly Flexible Virtual Reality System. *Future Generation Computer Systems 14*, 3–4, 167–178.

ROHLF, J., AND HELMAN, J. IRIS Performer: A High Performance Multiprocessing Toolkit for 3D Graphics. In *Proceedings of ACM SIGGRAPH 94*.

TRAMBEREND, H. 1999. Avocado: A Distributed Virtual Reality Framework. In *Proceedings of IEEE Virtual Reality '99 Conference*, IEEE Computer Society, 14–21.