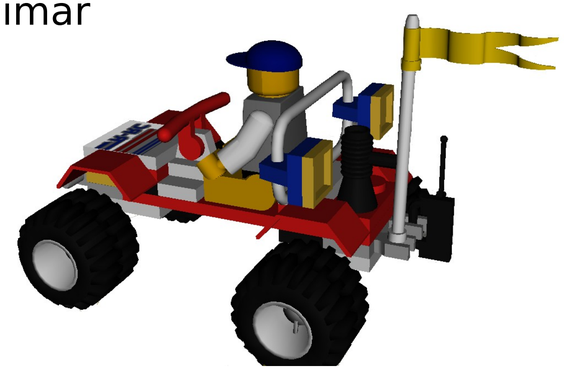
HECTOR
Scripting-Based VR System Design

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

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



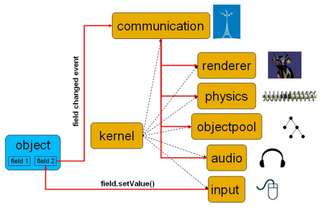
Abstract

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.

Module-Based Design

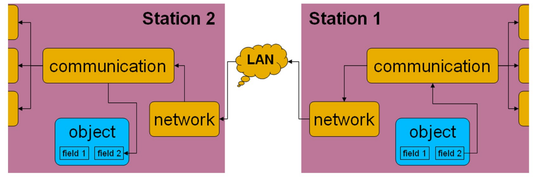
Individual HECTOR modules are threads that implement functionality for rendering, 3D sound synthesis, event distribution, physical simulation, and input device management. Most of these modules utilize existing libraries such as OpenGL|Performer [Rohlf and Helman], OpenSceneGraph (www.openscenegraph.org), Open Dynamics Engine (ODE, www.ode.org), or Spread (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 a communication module are initialized. All other modules are loaded upon request by newly created objects in the virtual environment. 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 sent to all distributed components that have been registered in the communication module for this event type.

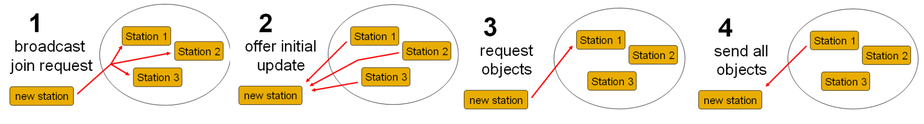


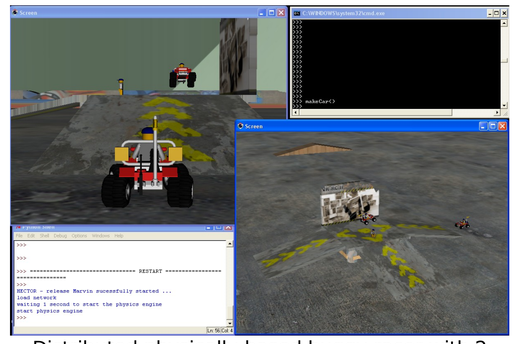
Networking

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.



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 using reliable TCP connections. Updates are multicast with unreliable UDP messages. 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 locally.





Distributed physically-based buggy racer with 2 participating staions on a single computer.

Villiard game controlled by tracked props.

Discussion

HECTOR is a lightweight, open source, scripting-based, distributed, and platform independent VR system. It is written in PYTHON and glues together 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 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 Proc of IEEE VR 2001, pp. 89–97.

Blach, R., Landauer, J., Rosch, A., and Simon, A. 1998. A Highly Flexible Virtual Reality System. Future Generation Computer Systems 14, 3–4, pp. 167–178.

Rohlf, J., and Helman, J. 1994. IRIS Performer: A High Performance Multiprocessing Toolkit for 3D Graphics. In Proc of ACM SIGGRAPH.

Tramberend, H. 1999. Avocado: A Distributed Virtual Reality Framework. In Proc of IEEE VR '99, pp. 14–21.