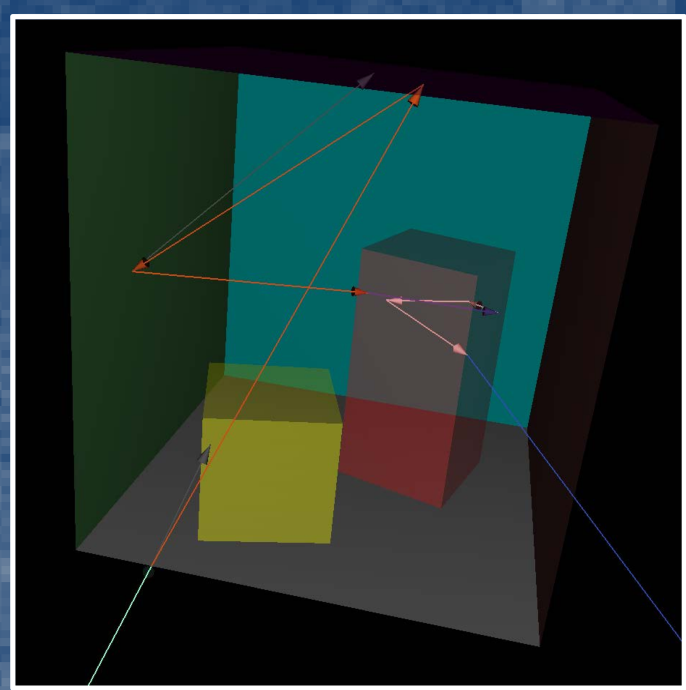
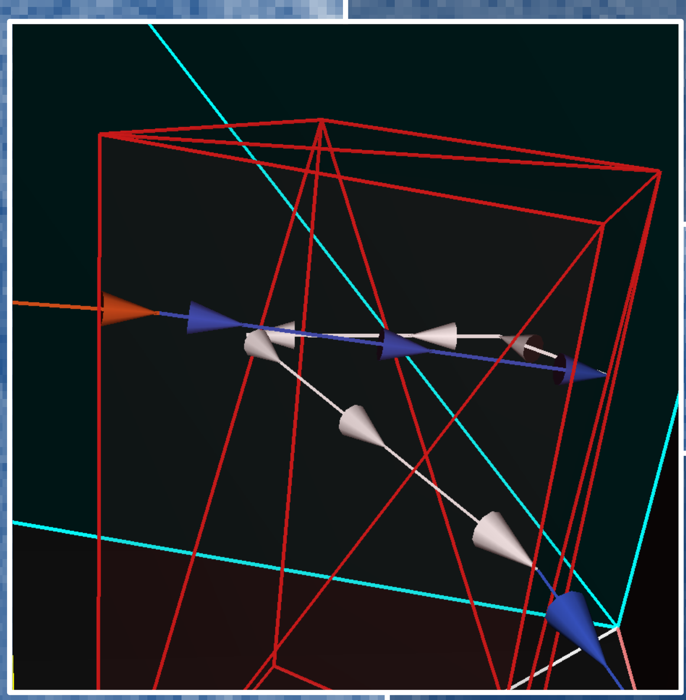
Ray Tracing Visualization Toolkit

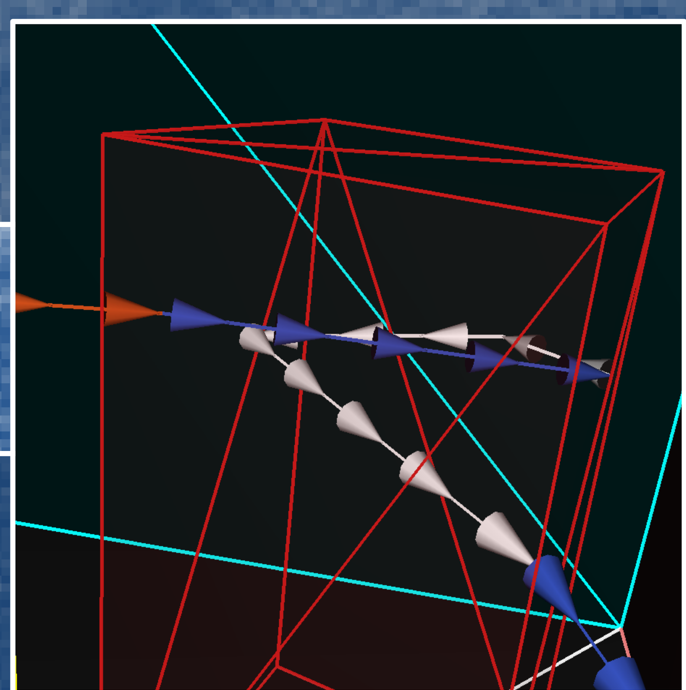
Jeremy Fisher, Daniel Eby, Ed Quigley, Gideon Ludwig, & Christiaan Gribble
Department of Computer Science
Grove City College
{fisherjf1,ebydj1,quigleyej1,ludwigga1,cpgribble}@gcc.edu

1. Introduction

Monte Carlo rendering algorithms typically simulate light transport throughout an environment via ray tracing. Even with recent advances targeting highly parallel platforms, high-quality images often require many seconds of computation to converge. The visual analysis of ray tracing operations may promote a better understanding of the way in which computation proceeds, thereby leading to insights that enable hardware resources to be utilized more effectively. This work-in-progress introduces the Ray Tracing Visualization Toolkit (rtVTK), a collection of C++ libraries and an extensible graphical user interface designed to support the visual analysis of ray-based rendering algorithms.







The following pseudocode demonstrates the use of core `rl` functionality in a basic recursive ray tracer:

```
// loop over pixels
for (uint y = 0; y < height; ++y)
  for (uint x = 0; x < width; ++x)
    // generate visibility ray and trace
    rlBeginTree(x, y);
    trace(visibilityRay, ...);
    rlEndTree();

trace(const Ray& r, ...)
  // perform ray tracing computations and recurse
  rlAddRay(r.o, r.d, r.t, ray.type, &my_data, sizeof(MyData));
  rlDescendTree();
  trace(nextRay, ...);
  rlAscendTree();
```

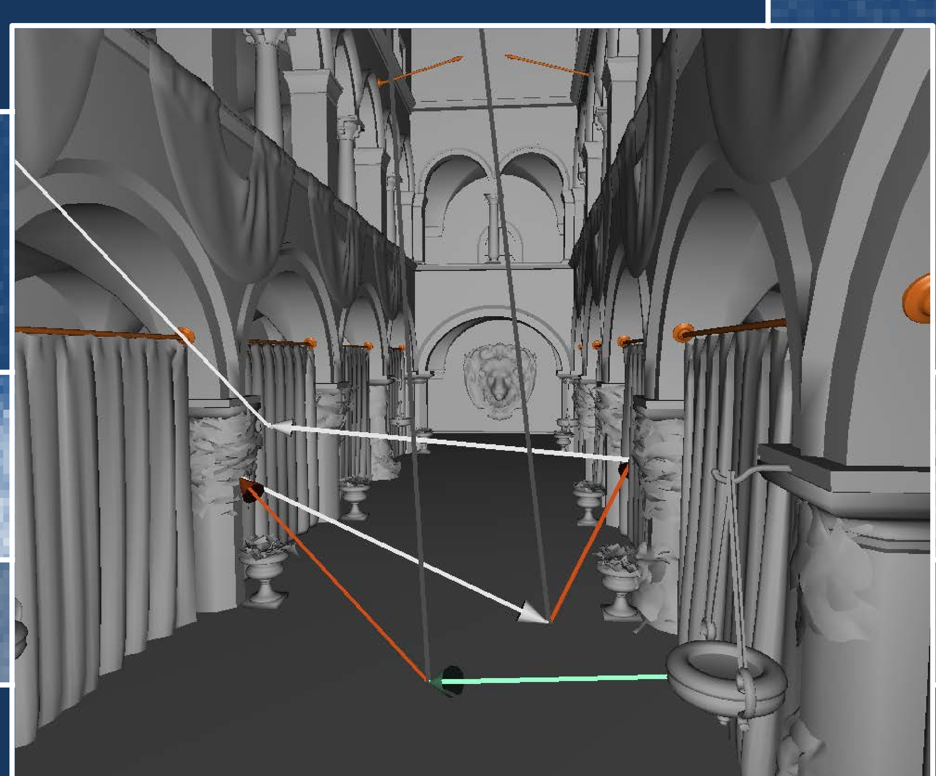
2. Components

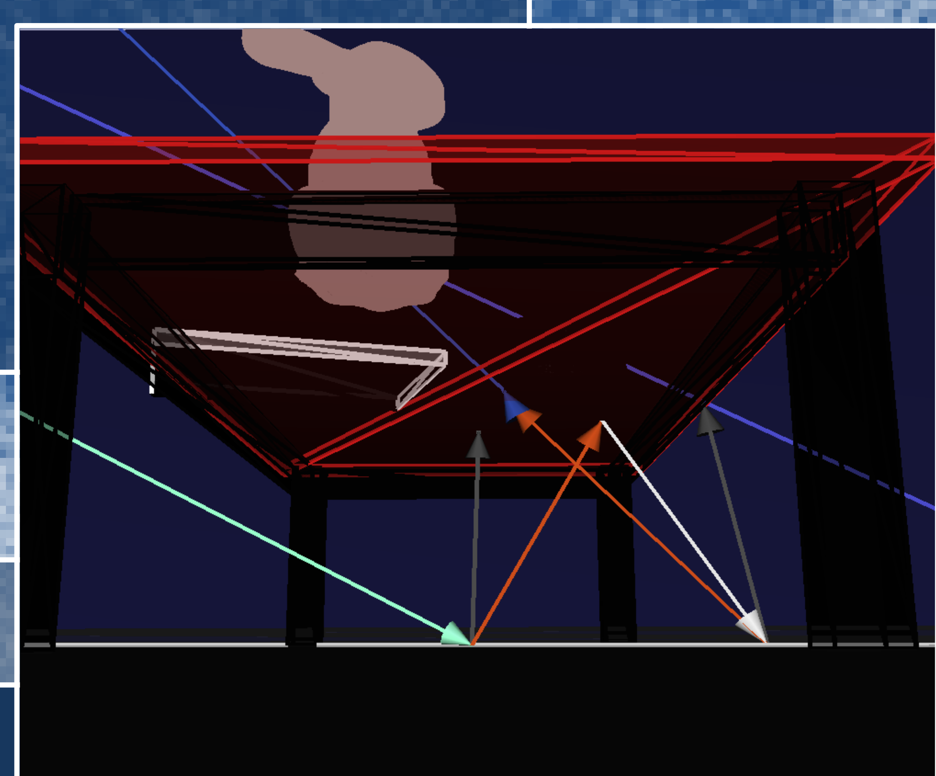
rtVTK consists of several components that combine to form a complete ray tracing visualization framework.

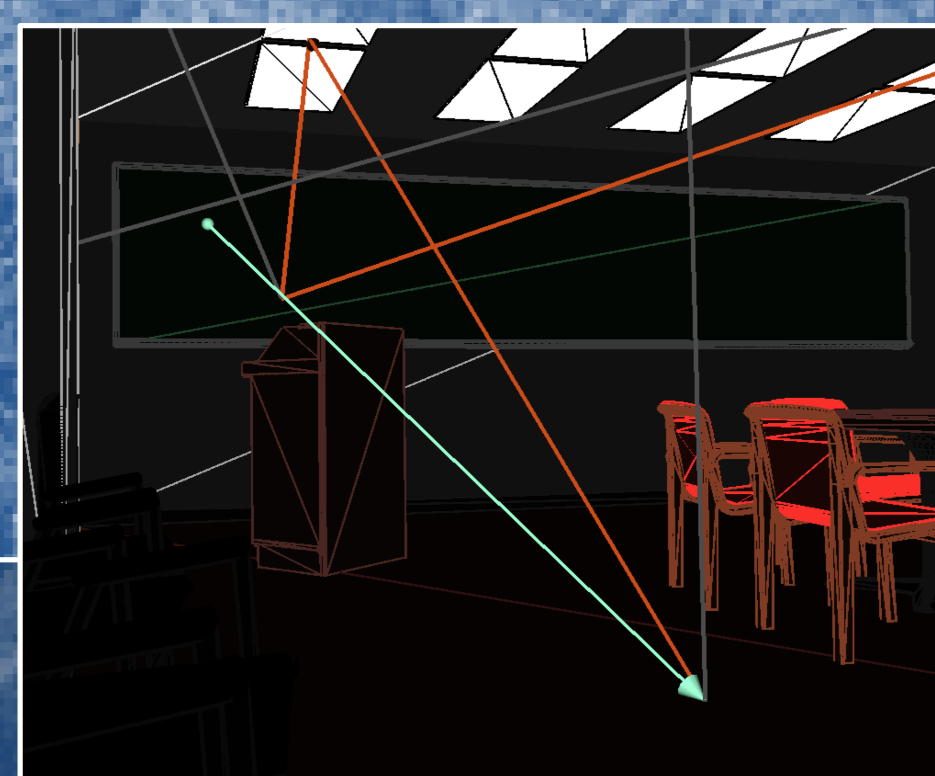
Rendering state recording library. An OpenGL-style API called `rl` captures rendering state in client applications. Existing renderers are instrumented with calls to the `rl` engine, and per-ray data—including arbitrary client payloads—is recorded throughout rendering. The resulting state is then explored using the rtVTK visualization engine or by layering additional components above the existing engine via plug-ins.

`rl` currently supports three modes of operation: *immediate* mode, for on-line renderers that export the rtVTK plug-in interface; *write* mode, for clients that capture rendering state for later processing; and *read* mode, for data visualization and other post-processing tasks. Common high-level operations such as ray tree traversal are implemented easily with `rlut`, a collection of utility functions that aggregate low-level `rl` operations. Finally, wrapper classes provide C++ bindings for seamless integration with object-oriented rendering clients.

rtVTK visualization engine. An interactive GPU path tracing plug-in and OpenGL/`rl` visualization plug-ins provide core rendering and visualization facilities. The rtVTK plug-in manager enables additional renderers and visualization components to extend the core facilities in a flexible, highly configurable manner. Finally, these elements are integrated with an extensible graphical user interface to provide a common ray tracing visualization process across multiple platforms, including Windows, Linux, and Mac OS X.







3. Discussion

Monte Carlo light transport algorithms must process billions of rays to generate highly realistic images. Visual analysis of the resulting state may lead to insights that enable more efficient implementation of these algorithms. The Ray Tracing Visualization Toolkit is designed to support these goals.

As the rtVTK framework matures, a number of interesting rendering and visualization problems—for example, new methods for real-time Monte Carlo ray tracing and new techniques to meaningfully represent large quantities of ray tracing program state—can be explored both thoroughly and rapidly.

Acknowledgments

This work was funded by the II-VI Foundation and the Grove City College Swezey Scientific Research and Instrumentation Fund. The GPUs used in this research were generously donated by NVIDIA through their Professor Partnership Program.