The Direct-Trace Library: Ray-Tracing for the Masses

Benjamin Mora
Swansea University
Swansea SA2 8PP, UK
b .mora@swan.ac.uk

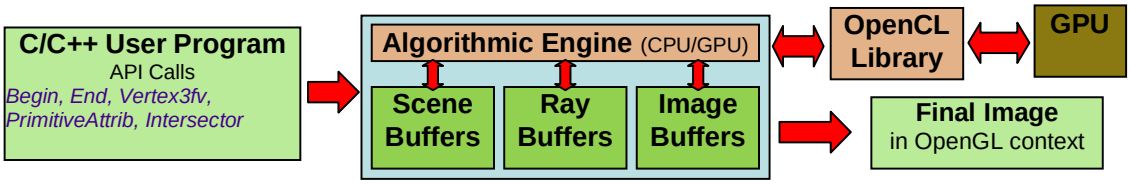

Figure 1: Overview of the current Direct-Trace Graphics library. The API stores Rays, Images, and Scenes as buffers and allows easy access to them through the interface. OpenCL acceleration is also available.

Abstract

We present the Direct-Trace API, a fast C/C++ ray-tracing library that does not employ spatial subdivision data structures for computing intersections. The library provides most of the tools needed in a rendering pipeline and keeps productivity high. This talk introduces the API and its features to the community.

Introduction

The Direct-Trace API is a Ray-Tracing library, and provides some specific atypical features, with an OpenGL-like interface for describing the scene. The Ray-Tracing engine runs the first efficient algorithm not to use spatial subdivision data structures. This allows tracing of rays as soon as the scene primitives are stored in appropriate lists, without needing any precomputations prior to tracing. On a single 3GHz core, the engine traces between 1 and 10 millions rays per second [Mora], depending on both the coherency of rays and scene size, which can contain several million primitives. Typical use of the library can include rendering either dynamic scenes or very large scenes where the scene content is streamed from an out-of-core source.

Unlike other Ray-Tracing platforms providing a high-level language and hiding most internals from the programmer, the new Direct-Trace library has been designed as a closer-to-hardware-but-programming-friendly layer. Principally, the library manages memory objects like buffers of rays, images and scene elements such as its geometry, and computes intersections. Other elements of the rendering pipeline such as shading or ray-generation can be treated with or without support from the library. For efficiency reasons (large batches of workload must be specified), the size of both ray and image buffers can/must be controlled by the programmer. Shaders can be loaded as C functions and/or OpenCL routines, but a higher-level language, sitting on top of the current software layer provided by our API, could be implemented by a third party.

OpenGL-like Scene Description Interface

The OpenGL interface is a standard well-known by programmers. By providing a similar interface, OpenGL programmers should be able to port their code easily. Porting will however require modifications in the source code, as some aspects of the OpenGL interface –a state machine– can incur a loss of productivity, and have therefore required some profound modifications as the library is designed to keep productivity high. For instance, resource allocation for rays, images and scenes is as simple as declaring C++ objects of those types. With scenes, programmers can explicitly define a given size for a scene (i.e., number of primitives), or choose to let the library automatically resize the arrays when adding elements to the scene, which will result in extra random allocations made.

Our major issue however, was to move from a rasterization concept, where only part of the scene needs to be known at a given time *t*, to a Ray-Tracing library where most information is needed throughout the pipeline (e.g., geometry). Indeed the OpenGL machine can process primitives of one given material sequentially, and then process primitives of another material type of anisotropic properties. As such, the OpenGL language would require extensions .

To solve this dilemma, the library mainly requires creation of materials with fixed-size properties enforced for any primitives of a given type. Streaming the scene geometry then requires fixed material in a Begin/End section. The following code specifies a triangle with normals as per-primitive attributes and computes intersections of a set of rays with the scene (shading left apart):

```
#include "DirectTraceAPI.h"
DirectTraceAPI dtAPI;
DTScene scene(dtAPI);
DTRayBuffer rays(dtAPI);
int byteMat=0, bytePrim=9*4, byteVertex=0;
int materialId= scene.NewMaterial(bytesMat,
                  bytesPrim, bytesVertex);
float vertices[3][3]={...};
float normals[3][3]={...};
scene.Begin(DT_TRIANGLES,materialId);
scene.PrimitiveAttrib(normals,9*4);
scene.Vertex3fv(vertices[0]);
scene.Vertex3fv(vertices[1]);
scene.Vertex3fv(vertices[2]);
scene.End();
rays.Resize(640,480);
rays.GenerateRaysFromGLProjectionMatrix();
scene.Intersector(rays);
```

Other Features and Current State

The library also provides multiple tools, including Ray and Image shaders, Multi-threading support, and OpenCL/OpenGL interoperability. Shaders specified with a C function pointer benefit from accessing the computer's main memory and implement all kinds of functionalities, while OpenCL shaders benefit from hardware acceleration, but may require more effort from the programmer.

While the features previously described are implemented and stable, the library is still under development. The library will be released by Siggraph 2011, with texture support, multithreading and 1.0 specifications finalized at that time.

References

MORA, B. Naive Ray Tracing: A Divide-And-Conquer Approach. Accepted for publication, ACM transactions On Graphics.

This work was supported by EPSRC (www.directtrace.org).