Analysis of Ray-Tracing Workloads and Acceleration Structures: First Checkpoint

Nicolas Feltman

October 29, 2011

1 Project Goal

Ray-tracing would be essentially impossible without acceleration structures to facilitate quick pruning of large portions of the scene. In current practice, these structures are constructed with essentially no knowledge of how rays are distributed in a scene, and little is even known about the inherent cost of tracing rays. The intent of this project is to examine how rays are typically structured in a scene, especially those associated with reflections and large light sources, and to examine how this knowledge can be utilized, on- or offline, in the construction and traversal of acceleration structures, particularly bounding volume hierarchies. We seek not only a robust metric of the efficiency of a particular tree traversal under a certain workload, but a method of building efficient structure with some information about ray distribution.

2 Definitions

In this section, I present an initial formalization of the ray-tracing problem, which will help to precisely state goals later.

2.1 Rays and Results

Let a scene S be the set of primitive objects that a ray-tracer is expected to render. Let O be a collection of rendering options used, including the view, lights, textures, the ray-tracer implementation, and any random seeds. These two values (via the `Init` function) determine R, an indexed set of ray queries. These rays, along with the scene data, are passed to the `Trace` function, whose result is P, an indexed set of intersections. Since R and P tend to travel together, we define RP to be those two sets zipped together (see below), and we call it a ray-intersection set. Note that for all but the simplest ray-tracers, part of R is recursively determined by P , namely reection/refraction and shadow rays. To disambiguate this chicken-and-egg issue, we decompose R and P into several generations:

$$\begin{aligned} R_0 &= \texttt{Init}(S, O) \\ P_i &= \texttt{Trace}(R_i, S) \\ RP_i &= (r_j; p_j) : r_j \in R_i; p_j \in P_i \\ R_{i+1} &= \texttt{Next}(RP_i, S, O) \\ R &= R_0 \cup R_1 \cup \cdots \\ P &= P_0 \cup P_1 \cup \cdots \\ RP &= RP_0 \cup RP_1 \cup \cdots \end{aligned}$$

This formulation does not fully resolve a temporal paradox raised by adaptive sampling, in which the results of casting some of the rays in R$_0$ (along with a myriad of other variables) determines other rays in R$_0$. Nor does this formulation capture that the `Next` function is typically defined so that every element of Ri+1 is independent of all but one element of RP$_i$, allowing for depth-first evaluation. For the purposes of this analysis, both of these details can be ignored.

2.2 Acceleration Structures

We now extend the `Trace` operation to take an acceleration structure B and a traversal method M. Of course, the actual results of `Trace` are invariant to B; only the run time is affected.

$$B = \texttt{Construct}(S, \ldots)$$
$$P = \texttt{Trace}(R, S, B, M)$$

It is worth noting that the particular acceleration structure B is, in general, restricted to traces over S. This bound relationship is understood and implied from here on.

2.3 Measuring Rays

Of course, all sorts of functions can be defined on any of the variables presented. I present a few examples here that will be used in the course of understanding the structure of rays in the scene.

2.3.1 Traffic(RP , H)

The Traffic function is a measure of the number of rays from RP passing through the region of space H. Other versions of this function weight the rays in some way, perhaps according to length.

2.3.2 Coherence(RP , H)

Coherence is a measure of the overall alignment of the rays in RP that pass through the region H.

2.3.3 TrafficField(RP)

The TrafficField function returns a scalar field for which the value at each point is the Traffic of RP in a region centered around that point.

2.4 Measuring the Structure

The following are a few examples of functions to measure the effectiveness of a BVH data structure.

2.4.1 Intersections(RP, B)

Intersections is a count of the number of intersections between the ray set RP and the bounding planes of the acceleration structure B. This can also be defined for un-intersected rays, R.

2.4.2 Traversal(rp, B)

Given a ray cast and a BVH, measure how many nodes exist on the intersection path in the BVH.

2.4.3 ProofCost(rp, B)

Similar to `Traversal`, measure the minimum cost of proving that a ray cast intersects a particular subnode and no others.

2.4.4 Cotraversal(rp1, rp2, B)

Given two particular rays, measure how many nodes of the data structure they both intersect.

2.4.5 CotraversalEntropy(RP, B)

Given an ordered set of ray casts, measure cotraversal as a function of distance between elements.

3 General Approach

Stated once again, the goal of the project is the analysis of ray tracing workloads and the effectiveness of common data structures in dealing with them. The goal is to derive and analyze real and theoretical costs of tracing various scenes, then showing how to build data structures and design traversal schemes to minimize this tracing cost, not only for average cases, but for known workloads.

4 Revised Schedule

The project is roughly on schedule. We present a very slightly update timeline.

4.1 Checkpoint 1 (October 28th) [DONE]

- Get a ray-tracer running for a few scenes and outputting all ray queries and results, including such data as ray type (generation). I plan to use the Intel Embree ray-tracer for this task as well as all later measurement collection.

- Build a tool for viewing and exploring a ray-intersection sets and the scenes from which they are formed. At a minimum, this iteration will include the feature to filter rays by generation. This tool will take as input the files that were output by the ray-tracer.

- Find a way to visualize the TrafficField function for a given scene.

4.2 Checkpoint 2 (November 18th)

- Write functions from section 2.4 to analyze the effectiveness of a particular BVH.

- Analyze and describe how current-practice build methods for acceleration structures distribute values like coherence or traffic among the data structure.

- Determine, at least in theory, the minimum amount of work to trace a scene.

- Derive a real-time cost function for the evaluation of `Trace`(R, S, B, M) in terms of R, S, and B and for a few traversal methods M. This will partly be driven by code inspection and modelling, but the end goal will be to fit curves to actual experimental runtime results. The prupose of this is to validate out model of the cost of tracing.

4.3 Endgame

At this point, the project can take one of two directions. Either we will have determined that current acceleration structure construction and traversal methods are adequate (or at least the problem is so intractable that there are no improvements to be made), or we will have found that those methods are inadequate. In the adequate case, we will attempt to show how BVHs have reached their theoretical limit. In the inadequate case, we will proceed as follows:

- Formulate a method for constructing and traversing an acceleration structure for a scene given a priori knowledge of all of RP.

- Adapt this method to work without the unrealistic assumption of full a priori knowledge of the results, but instead some sort of representative sample.

- Investigate further how the acceleration structure can be constructed at real-time speeds.

- Investigate ways to update the acceleration structure incrementally.