Advanced Graphics 2019/2020 – Assignment 2

Introduction

The goal of this assignment is to extend the framework you built in assignment 1 with acceleration structure construction and traversal. A successful implementation greatly reduces the cost of ray / scene intersection, and potentially enables real-time rendering using ray tracing.

BVH construction

The recommended acceleration structure for this assignment is the Bounding Volume Hierarchy (see “Freedom” below). An efficient BVH uses a compact node structure, aligned to cache lines to reduce memory traffic during traversal (see “Language Notes” below). Efficient construction is achieved by using a binning builder, as described by Wald [1].

BVH traversal

Traversing the BVH is a straight-forward process. A naïve implementation can be optimized using a custom stack rather than recursion, by using efficient ray/AABB and ray/triangle intersection code, and by traversing the BVH front-to-back (combined with ‘early out’).

Performance can be enhanced further using packet traversal. This can be implemented efficiently using the algorithms proposed by Overbeck et al. [3]. In C++, these algorithms benefit from (but do not require) SIMD / AVX; this may also be possible in C#.

Language Notes

This assignment may be executed in a programming language of choice. Support on the implementation side will be mostly limited to C++ and C# however, and performance is expected to be optimal for C++ code. Choice of programming language will not affect your grade.

Freedom

Most of the concepts discussed in the lectures apply to BVHs and kD-trees. You are free to use either data structure. You may also propose an alternative data structure (a ‘brickmap’ as in 5Faces perhaps?), if you wish to explore the specifics of that option. Please discuss your plans with me in advance.

You also have a certain level of freedom in the functionality you actually implement. Generally speaking, you can opt for excellence in BVH quality, construction speed or traversal speed to receive a high grade for this assignment.

Lighthouse

If you are working in Lighthouse, you have several options for handling geometry.

1. Easiest: import a single mesh (.obj file format) and illuminate it with point lights. This will send one mesh to via the RenderSystem to your Core (‘SetGeometry’). Build a BVH over this geometry and use this for ray/scene intersection. Assume the geometry is static.
2. Easy: have multiple meshes (multiple .objs, or a gltf file). This will trigger multiple calls to SetGeometry. Assume all geometry arrives before the first call to your Render method. Build the BVH over all received geometry during the first Render call (and *only* during the first render call). Assume the geometry is static.
3. Hard: build a BVH for each mesh you receive via SetGeometry. Using the matrices received via SetInstance, build a top-level BVH over the per-mesh BVHs. Update the top-level BVH each frame.

Practical Details

The deadline for this assignment is **Tuesday December 24, 23.59**. As usual, an extended deadline is available, at a 1pt penalty (it will also cost you Christmas, so please don't do this). The materials to submit are:

- your project, including sources and build instructions (if these are not obvious);
- a brief report, detailing implemented functionality, division of work, references and other information relevant to grading your submission.

You may work on this assignment alone, or with one other student. Feel free to discuss practical details on Slack/Discord. You are not supposed to share complete ray tracers there, but if everyone uses the same optimal ray/AABB test, that would be considered 'research'.

Tasks & Grading

A passing grade (6) for this assignment requires:

- implementing a correct BVH over a large (>10k) set of input triangles or other primitives;
- implementing ray/scene intersection using this BVH;
- achieving a performance improvement over brute force scene intersections in line with expectations.

To obtain an 8, chose one of the following:

1. Implement a combination of binned building and the surface area heuristic to determine good locations for split planes [1,4].
2. Construct a BVH for dynamic scenery using specialized builders for various types of animation. Add a top-level BVH to combine the resulting sub-BVHs, and adapt your traversal code to handle rigid motion.
3. Construct the BVH for a 100k triangle scene in less than 1 second.
4. Implement packet traversal for primary and secondary rays [3].
5. Construct a 4-way BVH by collapsing a 2-way BVH, and traverse this structure. The resulting traversal speed must be an improvement over 2-way BVH traversal.
6. Construct a high quality BVH by implementing the SBVH construction algorithm, including "unsplitting". Details are in the paper [2]. Warning: this is hard.
7. If you have something specific that you want to work on but it is not covered by 1..6, please talk to me.

Note: not all topics have been discussed in the lectures, and some require some research.

To obtain a 10, pick at least two from the list.

Purpose

We will use the result of this assignment in the third assignment. Regardless of the rendering algorithm you will be working on, fast intersection is crucial and allows you to work on something more interesting than a couple of spheres.

May the Light be with you,

- Jacco.

References

[1] On fast Construction of SAH-based Bounding Volume Hierarchies, Wald, 2007

[2] Spatial Splits in Bounding Volume Hierarchies, Stich et al., 2009

[3] Large Ray Packets for Real-time Whitted Ray Tracing, Overbeck et al., 2008

[4] Heuristics for Ray Tracing using Space Subdivision, MacDonald & Booth, 1990