Building a Hybrid Approach to Space Division for Faster Raytracing

Christopher Zhang (czhang44@binghamton.edu) | Kenneth Chiu (kchiu@cs.binghamton.edu) | SUNY Binghamton University

Abstract

Raytracing models the natural phenomena of light to create photorealistic images. Effects like shadows, reflections, and textures are produced with geometric equations. The majority of the computational work to generate raytraced images is spent determining ray-object intersections. Partitioning the space of the scene model has contributed to the speedup of raytracing (firing rays and finding the closest object intersection). The partition of space allows the intersection calculations to be performed more quickly by restricting the search space of possible collisions. However, making such a partition has a trade-off in performance versus cost of building such a structure. Scenes are highly variable in features like sparseness so there is no one superior method to partition a three-dimensional space to optimize raytracing. In this paper, I investigate using a uniform grid to slice up the three-dimensional space. This method does not slice the grid into boxes all at once but by splitting existing boxes iteratively. At each step, each box is sampled with a few rays to determine sparseness of the objects it contains. Based on a heuristic, that single box may be turned into its own bounding volume hierarchy or adaptive structure. I compare this hybrid approach to simply using a bounding volume hierarchy versus a uniform grid. A range of scenes are used to explore worst and average case scenarios. Performance is shown to be faster in the hybrid approach of not simply using a generic space division scheme.

Background + Related Work

Uniform grids rely on a 3-dimensional analyzer to determine which cells of a grid are crossed by a ray through its enclosed space. Partitioning the space is conceptually simple but the method of their construction leaves room for optimization and varying speeds. The recursive nature of a grid and a hierarchy to contain more grids and hierarchies has given rise to a few hybrid methods of construction. Hand (2012) proposes using regular subdivision with uniform grids (good build time, poor performance) that tries to quickly build a grid, tests its performance, and then determines whether to maintain the grid or use hierarchical space division (as with a BVH). The quantifier for performance includes a metric for sparseness as # empty cells / # cells. Because grids are better for uniformly distributed scenes, sparseness is not ideal when employing grids. Fellner (2000) proposes breaking a scene recursively into a hierarchy and using a cost function based on the number of primitives referenced by each cell to decide if the clusters of primitives that might be better suited for a BVH instead.

If the grid has a cost function defined on the assumption that the surface area/ratio can be used to determine the likelihood a ray will intersect a bounding volume given that it intersects its enclosing bounding box space. This method does not slice the grid into boxes all at once but by splitting existing boxes iteratively. At each step, each box is sampled with a few rays to determine sparseness of the objects it contains. Based on a heuristic, that single box may be turned into its own bounding volume hierarchy or adaptive structure. I compare this hybrid approach to simply using a bounding volume hierarchy versus a uniform grid. A range of scenes are used to explore worst and average case scenarios. Performance is shown to be faster in the hybrid approach of not simply using a generic space division scheme.

Procedure

1. Read primitives and scene description
2. Begin a binary tree BVH construction by partitioning along a pivot of a chosen axis (can cycle between x, y, and z)
3. For each BVH half, find the minimum and maximum extent
4. These are potential uniform grids of their own
5. Fire a set number of rays through each half node proportional to the volume of the bounding box formed by the minimum and maximum extent
6. Use the data returned (but not the rays hit or missed), this can be used to decide if that node is uniform or otherwise
7. Calculate the surface area of the bounding box and use it to disperse various ray origins that point across the box to other points that may be used as ray origins (this should be done randomly with different types of sampling to simulate the various rays that will be shot in a real-life model whether by the camera or with reflected light and shadows)
8. If it is not uniform, continue creating a BVH with that half and then consider later until the objects in consideration form a tight enough space to be split into a uniform grid
9. Once all nodes of a BVH is a uniform grid, the code need not be changed if the BVH and uniform grid both support a similar API (either through C++ polymorphism or other language features) as the only data to be returned is an intersection object with an Object pointer and a double distance to store how far away the intersection occurred

Algorithm

1. Use a BVH as a performance benchmark (rotating median axis split construction)
2. Use a UG to subdivide space efficiently and traverse using 3D differential analyzer
3. Adapтивely decide between using either BVH or UG during scene construction by sampling rays
4. Compare results of three methods of various amounts of primitives and resolutions

Results (1m pixel resolution)

<table>
<thead>
<tr>
<th>Scene</th>
<th>Triangle Vertices</th>
<th>Triangle BVH Create (ms)</th>
<th>BVH Tracing (s)</th>
<th>UG Create (ms)</th>
<th>UG Tracing (s)</th>
<th>Hybrid (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bunny1</td>
<td>3851</td>
<td>2.0386</td>
<td>3.719</td>
<td>1.4974</td>
<td>2.477</td>
<td>1.5927</td>
</tr>
<tr>
<td>Bunny2</td>
<td>69451</td>
<td>1.9320</td>
<td>73.814</td>
<td>2.0371</td>
<td>29.6</td>
<td>2.6387</td>
</tr>
<tr>
<td>Horse1</td>
<td>1850</td>
<td>1.9957</td>
<td>1.267</td>
<td>1.9014</td>
<td>2.83</td>
<td>2.1479</td>
</tr>
<tr>
<td>Horse2</td>
<td>96966</td>
<td>2.0000</td>
<td>114.927</td>
<td>2.7206</td>
<td>38.235</td>
<td>4.4191</td>
</tr>
<tr>
<td>Goldfish1</td>
<td>1400</td>
<td>1.9635</td>
<td>1.026</td>
<td>4.3683</td>
<td>1.843</td>
<td>5.1928</td>
</tr>
<tr>
<td>Goldfish2</td>
<td>22048</td>
<td>1.9885</td>
<td>22.981</td>
<td>5.0203</td>
<td>9.638</td>
<td>6.2164</td>
</tr>
<tr>
<td>Dragon</td>
<td>871414</td>
<td>1.9911</td>
<td>-</td>
<td>314.529</td>
<td>-</td>
<td>8.1181</td>
</tr>
</tbody>
</table>

Controls variables

- Scene descriptions
- Primitives (triangles can generalize most meshes)
- Cameras
- Lights

Independent variable

- Implementation methods of dividing space to speed up scene object queries and intersection calculations
- Time spent creating the subdivision structure
- Time spent drawing and tracing the rays to produce image

Dependent variable

- Number of primitives referenced by each cells
- Clustering clumps that...
- Volume of possible collisions

Works Cited

- "Efficient ray tracing", Solchen Bubble, 2006
- "In-field Scene Restructuring with Application to Ray Tracing", Fellner, 2003
- "AVR: An Accelerated Ray Tracing System", Fujimoto, 1986
- "When It Moves Sense to Use Uniform Grids for Ray Tracing", Hand, 2011
- "Ray Tracing Complex Scenes", Kajiya and Kojima, 1990
- "Physically Based Rendering: From Theory to Implementation", Peltz, 2010
- "Ray Tracing from the Ground Up", Saffari, 2007
- "An Efficient and Robust Ray-Box Intersection Algorithm", Williams, 2005