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, reflectance, and textures are produced with geometric equations. The majority of the computational workload to generate raytraced images is spent determining ray-object intersections. Partitioning the space of the scene model has contributed to the speed-up of rayshooting (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 factors 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-dimension differential analyzer to determine which cells of a grid are crossed by tracing a ray through its enclosed space
- Bounding volume hierarchies are 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
- Hapala (2011) 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
  - Variance in the number of primitives referenced by each cells alludes to the clustering clumps that might be better suited for a BVH instead
- Because grids are better for uniformly distributed scenes, sparseness is not ideal when employing grids
- Fellner (2000) proposes breaking a scene iteratively into a hierarchy and using a cost-based function to iteratively see if nodes should be turned into grids, which then have each cell considered if they should be turned into grids (if there is a disagreement at any point, a node will stay or transform into a BVH)
- Fellner (2000) bases the cost function used on the assumption that the surface areas' ratio can be used to determine the likelihood a ray will intersect a bounding volume given that it intersects its enclosing bounding box

Variables

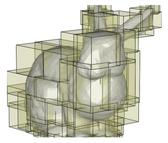
Controlled variables	Independent variable	Dependent variable
• Scene descriptions • Primitives (triangles can generalize most meshes) • Cameras • Lights	• Implementation methods of dividing space to speed up scene object queries and intersection calculations	• Time spent creating the spatial subdivision structure • Time spent drawing and tracing the rays to produce image

Results (1m pixel resolution)

Scene	Triangles	Triangle / Vertices	BVH Create (ms)	BVH Tracing (s)	UG Create (ms)	UG Tracing (s)	Hybrid (s)
Bunny1	3851	2.0386	3.719	1.49174	2.477	1.5927	1.4325
Bunny2	69451	1.9320	73.874	2.0371	29.6	2.6367	2.3412
Horse1	1850	1.9957	1.267	1.9014	2.83	2.1479	1.1349
Horse2	96966	2.0000	114.927	2.7306	38.235	4.4191	2.5633
Goldfish 1	1400	1.9635	1.026	4.3663	1.843	5.1928	4.5594
Goldfish 2	22048	1.9885	22.981	5.0203	9.638	6.2164	5.0034
Dragon	871414	1.9911	-	-	314.529	8.1181	6.7891

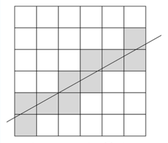
Procedure

Step 1



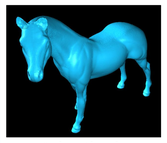
Use a BVH as a performance benchmark (rotating median axis split construction)

Step 2



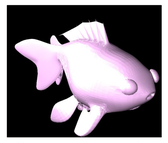
Use a UG to subdivide space efficiently and traverse using 3D differential analyzer

Step 3



Adaptively decide between using either BVH or UG during scene construction by sampling rays

Step 4

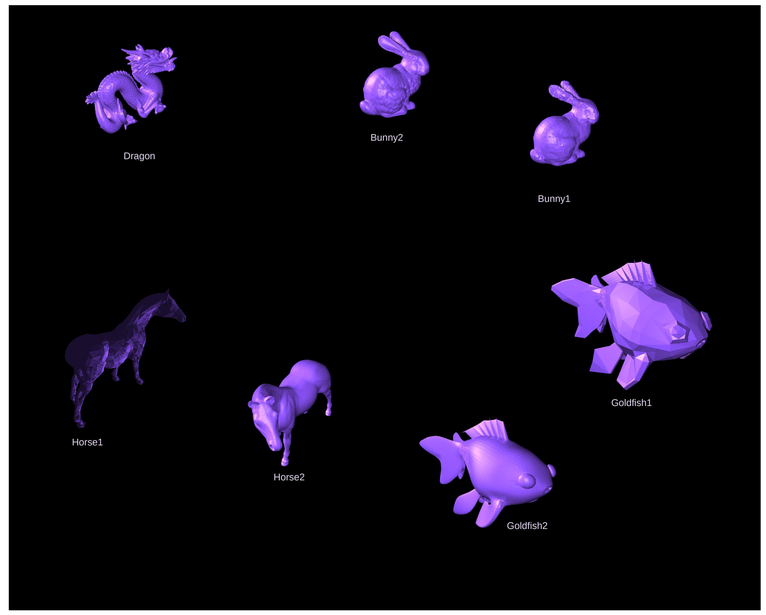


Compare results of three methods over various amounts of primitives and resolutions

Algorithm

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. Using the data returned (whether or not the rays hit or missed), use this proportion 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 one half 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

Scenes



Further Work & Conclusion

- From these results, the effort to adapt the interpretation of space to fit the current scene's qualities will benefit rayshooting, especially when scenes are neither consistently nor evenly dispersed
- There are a few different ways to account for sparseness based only on viewing the primitives present in the structure (shooting rays randomly and sampling the structure to determine uniformity requires user parameters that are suboptimal and can be bettered)
- While results for the hybrid are not always best overall for all scenes, it does not tend to be as slow as sticking to either one or the other structure in any case
- Being smart about choosing a spatial subdivision structure requires processing overhead and should be as simple as possible for the process to be automatic, hence heuristics are the most appealing methods
- Further work in this effort can be directed into using more modern BVH construction methods (insertion-based, bottom-up methods) in addition to less simplistic gridding techniques such as tetrahedral units rather than cubes to tessellate 3D space
- Raytracing is only viable when the underlying algorithms are performant so the effort to improve what seems to be only possible by brute force tracing light in infinite directions to generate an image will be a continuous one

Works Cited

- Notes on efficient ray tracing, Solomon Boulos, 2008
- Hybrid Scene Restructuring with Application to Ray Tracing, Fellner, 2000
- ARTS: Accelerated Ray Tracing Systems, Fujimoto, 1986
- When It Makes Sense to Use Uniform Grids for Ray Tracing, Hapala, 2011
- Ray Tracing Complex Scenes, Kay and Kajiya, 1986
- Physically Based Rendering: From Theory to Implementation, Pharr, 2010
- Rethinking Graphics and Gaming Courses Because of Fast Ray Tracing, Shirley, 2007
- Ray Tracing from the Ground Up, Suffern, 2007
- Spatial Splits in Bounding Volume Hierarchies, Stich, 2009
- State of the Art in Ray Tracing Animated Scenes, Wald, 2009
- An Efficient and Robust Ray-Box Intersection Algorithm, Williams, 2005