Efficient Acceleration Data Structures for Real-Time Ray Tracing

Aaron Lemmon
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
lemmo031@morris.umn.edu

ABSTRACT

Needs more work

Ray tracing a single frame of a 3D scene can involve testing for intersections of billions of rays against millions of primitives such as triangles. In order to speed up this process, acceleration data structures can be used to organize the primitives by their location so that only a small subset of them need to be tested for intersection against any given ray. Acceleration data structures usually take the form of a tree: the top node represents the entire 3D volume of the scene and the children of every node divide up the volume of their parent node into subsections. Although these acceleration data structures speed up ray intersection, the time it takes to build these trees can negatively impact performance. This is especially apparent in scenes with moving objects since the acceleration data structures need to be rebuilt or updated to accurately reflect the new locations of objects.

This research addresses methods for building and maintaining acceleration data structures in a way that minimizes the combined time spent constructing the data structure and using it to test for ray intersections. In particular, parallel computing can be effectively utilized to decrease the time spent on an acceleration data structure. More efficient acceleration data structures allow for ray tracing scenes with motion in real time.

Keywords

thing1, thing2, thing3

Needs more work

1. INTRODUCTION

Sources here.

[2] [1]

Needs more work

UMM CSci Senior Seminar Conference, May 2016 Morris, MN.

2. BACKGROUND

In 3D computer graphics, objects are made up of a collection of *(*primitives), which are usually simple geometric shapes like triangles. A 3D *scene* consists of all the primitives that construct it. In order to depict a scene on a display, the pixels of the display must be colored to create an image. A technique called *ray tracing* can be used to color the pixels in a way that can accurately portray shadows, reflections, and refractions in a scene. Ray tracing achieves this by determining how light travels in a scene from the light sources, reflecting or refracting off objects, and meeting the viewer.

Since many rays of light from a light source may not ultimately reach the viewer, it is more practical to start from the viewer and trace paths of light backwards. For every pixel on a display, a ray is traced from the viewer through the pixel and into the 3D scene. When a ray intersects with an object in the scene it can recursively generate more rays in directions that will contribute to an appearance of reflections, refrations, or shadows. While ray tracing can create highly realistic images, the process of tracing a huge number of rays on displays with many pixels with 3D scenes with many primitives can take a very long time.

Needs more work

3. ACCELERATION DATA STRUCTURES

One of the most time-consuming aspects of ray tracing involves testing for intersections between rays and primitives in the scene. This is because there are usually very many primitives in a scene, and an intersection needs to be checked for all of them. However, if the primitives are stored in a data structure that helps identify where they are located in the scene, then a ray only needs to check for intersections with objects located in the parts of the scene the ray is passing through. This can drastically improve the performance of intersection testing for each ray.

A common way to make intersection tests easier to calculate is to surround primitive with bounding boxes. A bounding box is just a box that completely contains another object as tightly as possible. A common approach is to use axis-aligned bounding boxes (AABBs), which are aligned with the axes of the scene as a whole. It is usually much simpler to test for ray intersections with AABBs than with the objects they contain. If a ray does not intersect with an object's AABB, then it cannot intersect with the object itself. However, if it does intersect with the AABB, then a more costly check must be made to test for intersection with the contained object. Overall, the use of AABBs can reduce the cost of testing for intersections since a ray misses many more objects than it hits.

Bounding Volume Hierarchies (BVHs) extend the idea of AABBs to a tree-like data structure. The root node of a BVH is an AABB that contains the entire scene. The child nodes of any parent node subdivide the total volume of the parent node into multiple smaller sub-volumes. By continually separating volumes into smaller volumes, it becomes possible to group close primitives together. The lower nodes on the tree give more precise location information than the higher nodes.

Searching for a ray intersection with objects in a BVH occurs in a top-down manner. First, the ray is tested against the root of the node to check if it even intersects with the scene. If it does, then each of the root's children is checked for intersection. Any child node that the ray misses can be entirely eliminated from the rest of the search, since the objects contained entirely within the node will also not intersect. Ray tracing with a BVH structure can therefore greatly reduce the number of intersection checks performed per ray.

Needs more work

3.1 A Subsection

Needs more work: change header

4. PERFORMANCE COMPARISONS

Needs more work

5. CONCLUSION

Needs more work

Acknowledgments

Needs more work

6. REFERENCES

[1] K. Garanzha, J. Pantaleoni, and D. McAllister. Simpler and faster hlbvh with work queues. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, pages 59–64, New York, NY, USA, 2011. ACM.

[2] T. Karras and T. Aila. Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*, HPG '13, pages 89–99, New York, NY, USA, 2013. ACM.