1 Introduction

2 Background

2.1 Ray Tracing

2.2 Acceleration Structures

The naive way to trace a ray would be to test it against each primitive in the scene and report the intersection closest to the source of the ray. This is of course $O(n)$ on the number of primitive and would be completely infeasible for all but the smallest scenes. To overcome this complexity, the scene is carefully split into several smaller continuous parts, the union of which comprise the whole scene. Thus, rays are only tested against the primitives in a part of the scene should it be determined that the ray enters that part of the scene at all. Popular methods for splitting include octrees [], various grid-based systems [], kd-trees [], and bounding volume hierarchies []. Havran's PhD thesis [**?**] provides a good early comparison of these various schemes, although the landscape has changed considerably since them.

This project will focus exclusively on methods involving bounding volume hierarchies (henceforth referred to as BVHs). In a BVH, the set of objects in a scene are partitioned into two or more subsets, the exact number of which is called the branching factor, and a bounding volume is constructed around each subset. Each subvolume may be further split into smaller parts, thus forming a tree with groups of primitives at the leaves. There is no requirement that sub-volumes of a BVH not overlap, although minimizing this overlap is a goal in the construction of BVHs. Most BVH implementations use axis-aligned bounding boxes for bounding volumes (as the rest of this project will assume), because they are relatively easy to store and test for intersections [], while being sufficiently descriptive. Although initially found to be lacking [**?**], BVHs have been an area of intense research. Their relative permissive invariant not only makes them well-suited for applications in animation, but also allows for easy implementation of some packet-traversal algorithms [**?**].

2.3 BVH Traversal

The basic observation behind a BVH is that a ray will only intersect a primitive object if it intersects every bounding box containing that object in the hierarchy. Thus, if a ray does not intersect the bounding box describing a particular node in the hierarchy, all objects under that node can be discarded. For a single ray, this implies a simple traversal algorithm. Starting at the root of the BVH, test for intersection with the bounding box of the node. If one exists, recur for every subnode and pick the closest of the childrens' intersections. If the bounding box is missed, or if every child node is missed, then return a miss for the node. At leaves, simply test for intersection with every contained primitive.

2.3.1 Subnode Test Order

This optimization makes use of the fact that an intersection in the near part of the ray obviates the need for further tests of the rest of the ray. If the children of a node do not overlap, which is rare, they can be tested in front-to-back order from the point-of-view of the ray, so that the first intersection will be the true intersection for that node. If the children do overlap, then there are two algorithmic options. The simpler option is to test the children nodes depth-first, skipping subnodes only when their bounding boxes lie entirely behind the closest intersection so far. This benefits from a simple *a priori* ordering, as described in [Wald07]. The more complicated option is to traverse all nodes in front-first order, which requires maintaining an active-node list, as described in [**?**].

2.3.2 Packet Tracing

2.3.3 Coherence

2.3.4 Non-Packet Methods

2.4 BVH Construction

2.4.1 Surface Area Heuristic

2.4.2 Fast Parallel Build

2.4.3 Tree Rotations

2.5 BVH Updates for Animations

2.5.1 Tree Refitting

2.5.2 Incremental Tree Rotations

2.6 Using Ray Distribution Knowledge