Ray Tracing Acceleration Data Structures

Sumair Ahmed

October 29, 2009

Ray Tracing is very time-consuming because of the ray-object intersection calculations. With the brute force method, each ray has to be tested with all objects and then the closest intersection is obtained. For a scene containing |O| number of objects and a resulting image composed of |I| pixels, the complexity will be |I| x |O|. Different acceleration data structures are used and the important characteristics of it are construction time, memory use, and ray traversal time.

1. Spatial Data Structures

The Rendering space is divided within the environment instead of enclosing the objects in bounding volumes. These spatial subdivisions require to have elementary cells of the same shape and size. The constructed elementary cells are either addressed directly or from the hierarchical cells. The construction of spatial subdivisions usually proceeds in a top-down way. We shall continue below with the description of the most commonly used spatial subdivisions.

1. uniform grids
2. bounding volume
3. bounding volume hierarchy
4. hierarchical grids
5. octrees/quadtrees, BSP-trees, kd-trees

1.1 Uniform Spatial Subdivision

Space is subdivided by a regular grid for example,Uniform Grid. The Ray steps through the grid looking for primitives/objects as long as the grid cells/voxels are empty. So the objects close to the path of the ray are tested for intersection. The preprocessing of the scene before the ray traversal starts can be quite costly.

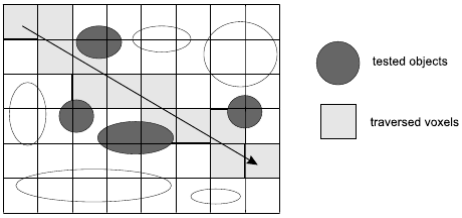

Figure 1: Example of an Uniform Grid-Ray Traversal tests only the blue colored grid cells for an Object Intersection

Since the uniform grid is created regardless of the occupancy of objects in the voxels, it typically forms many more voxels, and therefore it demands necessary storage space. Nevertheless, the ray traversal algorithm for the uniform grid can be performed very efficiently, since the voxels are of the same size. The ray traversal algorithm, is analogous to the Bresenham algorithm for drawing a straight line in 2-dimensional space and thus requires a simple operation for each traversal step.

The disadvantage of the uniform grid is that the occupancy of most voxels can be very small, particularly for sparsely occupied scenes. In this case a ray typically has to traverse many empty voxels before hitting the full voxel. The uniform grid does not adapt to the distribution of the objects in the scene.

1.2 Bounding volume

A naive Ray tracing algorithm tests every object for intersection with a given ray. The ray-object intersection test itself can be an expensive operation, particularly for some shapes of objects (Spline surfaces, polygons with many edges, etc.). Therefore it is advantageous to enclose tightly the object in a bounding volume with a simple

ray-object intersection test. Practically, the bounding volume of an object O is a cell V for which holds $O \wedge V \equiv O$. If a ray intersects the objects bounding volume, an intersection cell between the ray and the object is performed. If a ray does not intersect the bounding volume, it cannot intersect the object and thus a substantial part of the computation can be avoided on average. Sphere,Cube are the simple examples of a Bounding Volume.

1.3 Bounding volume hierarchy (BVH)

A bounding volume hierarchy is simply a tree of bounding volumes. Given the bounding volumes of objects, an n-ary rooted tree of the bounding volumes is created with the bounding volumes of the objects at the leaves. The bounding volume at a given node encloses the bounding volumes of its children.

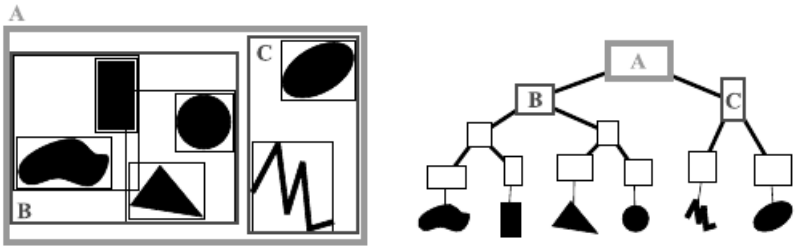

Figure 2: Example of a bounding-volume hierarchy, using rectangles as bounding volumes

The hierarchy provides naturally the method for testing a ray with the objects in the scene. If the ray does not intersect the enclosing bounding volume at the root node, it cannot intersect any object. Otherwise, the hierarchy is recursively descended again only for those nodes of BVH whose bounding volumes were intersected by the ray. Note that although a bounding volume of a node always completely includes its child bounding volumes, these child bounding volumes can mutually intersect.

The best bounding volume will be determined by the shape of the underlying object or objects. For example, if the objects are long and thin then a sphere will enclose mainly empty space and a box is much better. Boxes are also easier for hierarchical bounding volumes. Note that the intersection computational time using a hierarchical system is something between linear and a logarithmic dependence. This is because, for a perfect case, each intersection test would divide the possibilities by two, and we would have a binary tree type structure.

In addition, the cost of processing a node in the hierarchy is composed of the costs of accessing the location of the node in memory, the cost of reading the node's children pointers and their bounding volumes, and the cost of intersection or distance computations on those bouding volumes.

Hierarchical Grids

A hierarchical grid is a typical example of non uniform space subdivision where the space adapts to the scene, having bigger grid cells in the low density areas and smaller cells in the high density portions of the scene. It avoids the regularity of uniform grids, since the regularity is particularly inconvenient for sparsely occupied scenes. The common principle used in hierarchical grids is to insert uniform grids recursively into other uniform grids. The known methods of several hierarchical grids strongly differ in the construction phase. Hierarchical grids also require a more complicated ray traversal algorithm than uniform grids.

Octree/Quadtree

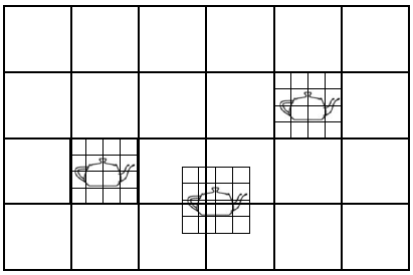

Figure 3: Hierarchical Grid

With an Octree, the 3 dimensional Space (ex:Cube) is subdivided into 8 equal parts. It is built recursively in top-down fashion, but unlike to the BSP tree(explained later) the initial cell is not split into two cells but into eight cubic cells. The position of the splitting plane cannot be specified as it has to be uniformly subdivided (shown in Figure 4).

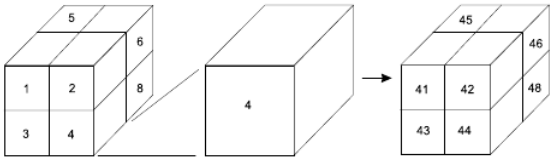

Figure 4: Octree Space partitioning

The octree can serve as the representation of a three-dimensional object. Here, the leaves of the octree are marked either empty or full. Cells with high object occupancy can be recursively subdivided into smaller and smaller cells, generating new cells in the octree. However this results in large empty cubes for empty scene areas, and many smaller cubes for detailed areas.

The ray traversal algorithm for the octree is more complicated than for the BSP tree or the uniform grid. Each ray intersecting an octree node can visit at most its four of eight descendants, and the computation of their order to be visited along the ray path is thus more involved than for the BSP tree. The time consumed by a ray traversal algorithm for the octree is given by the efficiency and robustness of the algorithm determining which child nodes are to be visited and in which order. A 2-dimensional axes representation of Octree, namely Quadtree, is shown in Figure 6. Quadree splits each square into 4 equal sized squares.

Binary Space Partitioning (BSP) Tree

A Binary Space Partitioning (BSP) tree is a spatial subdivision that can be used to solve a variety of geometrical problems. It was initially developed as a means of solving the hidden surface problem in computer graphics. It is a higher dimensional analogy to the binary search tree.

A BSP tree is usually constructed hierarchically in top-down fashion. At a current leaf V a splitting plane is selected that subdivides into two cells. The leaf then becomes an interior node with two new leaves. The objects associated with V are distributed into its two new descendants. The process is repeated recursively until certain termination criteria are reached. Commonly used termination criteria are the maximum leaf depth and the number of objects associated with the leaf. The BSP tree has two major variants in computer graphics, namely axis-aligned and polygon-aligned.

The polygon-aligned form chooses a plane underlying the polygon as the splitting entity that subdivides the spatial region into two parts. The scene is typically required to contain only polygons, which is too restrictive for ray shooting applications.

In the axis-aligned form of the BSP tree the splitting entity is the plane that is always perpendicular to one of coordinate axes. Since the splitting planes are perpendicular to the coordinate axes, the spatial subdivision is also called a rectilinear BSP tree or an orthogonal BSP tree.

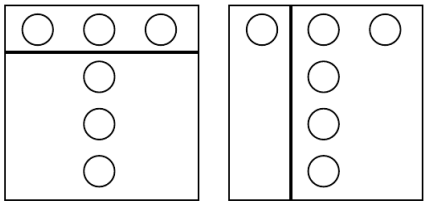

Figure 5: Median cut along an axis aligned plane of a BSP tree (a),(b)

The orthogonality of splitting planes in the BSP tree significantly simplifies the intersection test between a ray and the splitting planes. The cost of computation of the signed distance corresponding to the intersection point between a ray and the axis-aligned splitting plane is roughly three times lower than for an arbitrary positioned plane.

An important feature of the axis-aligned form of the BSP tree is its adaptability to the scene geometry that is induced by the possibility to position the splitting plane arbitrarily. Traditionally, the splitting plane is positioned at the mid-point of the chosen axis (shown in Figure 5), and the order of axes is regularly changed on successive levels of the hierarchy.

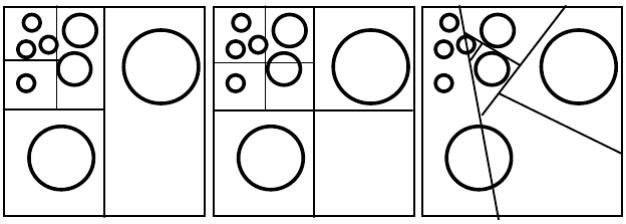

Figure 6: Non Uniform Subdivision using Kd-tree, Octree, BSP tree

Kd-Tree

This is a k-dimensional tree whose nodes divide along one of the Cartesian axes. It starts by splitting the scene bounding box along one axis. Next, split these halves using the next axis and recursively do this process. These seem to be the best choice at the moment for complex scenes. The position of the splitting plane can be specified, but it is usually chosen in a way that balances the number of primitives to the left and the right of the plane, while minimizing primitive splits.

Kd-trees are a variant of BSP trees where the splitting planes are only axis aligned. The positioning of the splitting plane is used to distinguish between the BSP tree and the kd -tree. Conceptually, the BSP tree and the kd -tree are equivalent. The important feature of the kd -tree is that it always has axis-aligned splitting planes unlike the BSP tree.

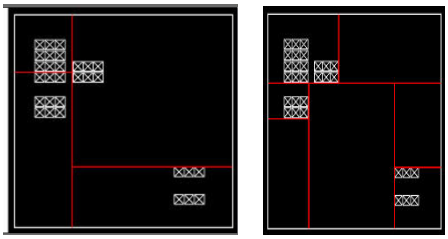

Figure 7: Kd-tree built using different rules

Splitting the objects well is what differentiates a good kd-tree from a bad one. In the left image (shown in Figure 7), scene is subdivided in such a way that the number of polygons on both sides of the plane is roughly the same. This is not a good Splitting tree. Imagine a ray traversing this scene, it will never pass through an empty voxel. If we keep adding planes to this tree using the same rule, we will end up with a tree that does not contain a single empty node. This is pretty much the worst possible situation, as the raytracer has to check all primitives in each node it travels through. In the right image, a different rule has been used to determine the position of the split plane. The algorithm tries to isolate geometry from empty space, so that rays can travel freely without having to do expensive intersection tests.

The properties of the octree that are related to the Ray tracing can be compared to the properties of kd -tree (or the BSP tree). However, there are cases where the performance of Ray tracing based on the octree and the Ray tracing based on the kd -tree significantly differ. The geometry induced by the octree can be simulated by the BSP tree, but the octree has a more regular structure. Depending on the ray traversal algorithm, the order of all the octants to be visited in the ray traversal algorithm is usually determined even if the ray can terminate already in the first octant. Small occupancy of the leaves for sparsely occupied scenes is the next disadvantage of the octree. The empty neighbor leaves in the octree have to be determined and traversed, whereas in the case of construction of the BSP tree it is likely they would be represented by a single leaf. The small occupancy of leaves implies relatively large memory requirements for octree representation.

References

- Bounding-Volume Hierarchy - *http* : *//www.cosy.sbg.ac.at/ held/projects/collision /bvt.html*
- Introduction to bounding volume hierarchies - *http* : *//www.win.tue.nl/~hermanh /stack/bvh.pdf*
- Index to Ray Tracing News - *http* : *//tog.acm.org/resources/RTNews/html /rtn_index.html*
- Heuristic Ray Shooting Algorithms - *http* : *//www.cgg.cvut.cz/members/havran/ phdthesis.html*
- RayTracing: Spatial subdivisions - *http* : *//www.devmaster.net/articles/raytracing_series /part4.php*