Ray Tracing Acceleration Techniques

Lecture #3:	Tuesday, 23 January 2003
Lecturer:	Greg Humphreys
Scribe:	Rui Wang
Reviewer:	Nolan Goodnight

1 Classification of Acceleration Techniques

Finding ray-object intersection is computationally expensive. Without acceleration (brute force), each ray has to be tested with all objects and then the smallest t value is obtained as the closest intersection. For a scene containing $|O|$ number of objects and a resulting image composed of $|I|$ pixels, the complexity will be $|I| \times |O|$. Ignoring super-sampling, reflection and refraction calculation, one million objects with 1024^2 pixels will result in one trillion ray-object intersection calculations. It's bad. Different data structures used to accelerate the computation. Below is a survey of ray tracing acceleration techniques by James Arvo and David Kirk. Notice that "generalized" rays are not always useful, because for them,

1. Intersection is not well defined or hard to find. For example, a bean ray intersects an arbitrary polygon.

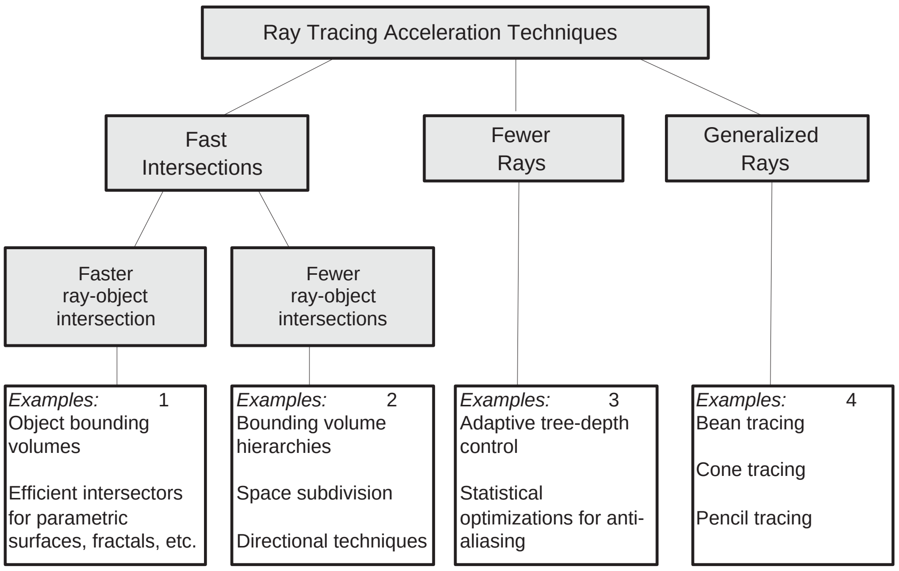

Ray Tracing Acceleration Techniques
Fast Intersections
Fewer Rays
Generalized Rays
Faster ray-object intersection
Fewer ray-object intersections
Examples: 1
Object bounding volumes
Efficient intersectors for parametric surfaces, fractals, etc.
Examples: 2
Bounding volume hierarchies
Space subdivision
Directional techniques
Examples: 3
Adaptive tree-depth control
Statistical optimizations for anti-aliasing
Examples: 4
Bean tracing
Cone tracing
Pencil tracing

Figure 1: A broad classification of acceleration techniques.

2. Ray is not closed under reflection from curved surfaces. For example, a cone ray intersects a reflective sphere.

Also notice that "Fast Intersections" is further classified as

1. Faster ray-object intersection, which involves in better optimizing code, finding bounding volume that tightly fits an object and is efficient for intersection check.
2. Fewer ray-object intersection, which won't speed up individual intersection, but will determine intersection of a ray with N objects in sub-linear time. We will focus on this technique.

2 Uniform Grids

2.1 Structure

Virtually all techniques involve spatial index structures. "Uniform grids" is one of the simplest and implemented in lrt. The steps are:

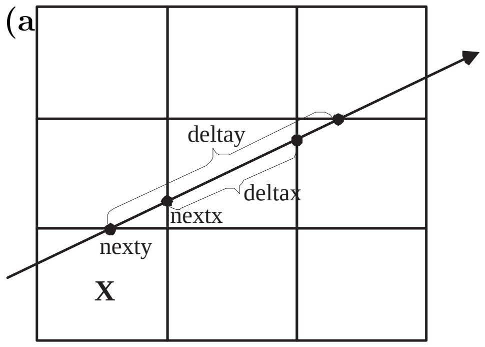
1. Compute bounding box (bbox) of the scene
2. Divide bbox into grids of certain resolution, which is often determined by: $n^3 = d|O|$. Where d is a constant (in lrt it's 8) and $|O|$ is the number of objects.
3. Place objects into cells by:
   - Calculating bbox of each object,
   - For each overlapped cell, put a pointer to the object in cell structure.

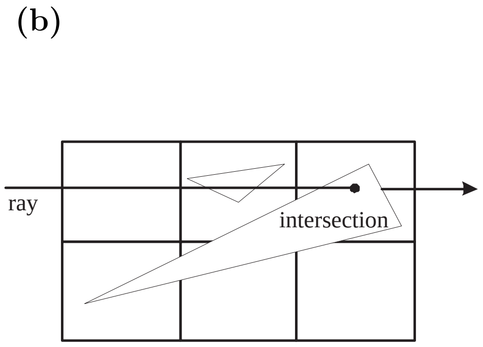
2.2 Traversal Algorithm

Once the uniform grid is built up, ray-object intersection is detected by running a traversal algorithm. It's similar to 3D DDA, as illustrated by Fig.2a:

1. Initialization. At ray origin (X, Y), compute: next ray intersection with horizontal edge *nextx* and vertical edge *nexty* (both are values of t); distances along the ray between horizontal crossings *deltax* and vertical crossings *deltay*,
2. Incremental traversal by running the following loop:

```
if(nextx<nexty)
   nextx += deltax, X += 1;
else
   nexty += deltay, Y += 1;
process_grid(X, Y);
```





(a) **(b)**

Figure 2: Traversal algorithm and possible problems

Notice that stepping for X and Y might be +1 or -1, depending on which direction the line goes.

Be careful when implementing this algorithm: (see Fig.2b)

- make sure the intersection is inside the current grid,
- the same ray-object intersection might be computed multiple times, one solution is to use "mailbox" to record the intersection between a specific ray and an object. "mailbox" is not always effective.

3 Spatial Hierarchies

3.1 Structure

The first example is tree-structure hierarchy such as K-d tree. Fig.3 illustrates 2-d tree, where an axis-aligned plane is introduced to break down each volume into two sub-volumes. The subdivision stops when the number of objects overlapping a cell falls below a certain threshold. Similarly, we can construct quadtree, octree, etc. In BSP tree, the partitioning plane is not necessarily axis-aligned. As in uniform grid, objects must be assigned to cells. Heuristics on where to put the partitioning plane include:

- Median-cut: find the plane that puts approximately equal number of objects at each side.
- Surface-area heuristics: the number of rays in a given direction that hits the object is proportional to the projected area A. (see Fig.4) And rays in all directions that hits the object is proportional to the surfaces area S. Crofton's Theorem states that $S = 4\overline{A}$, where $\overline{A}$ is average projected area in all directions. The proof can be found at Greg's hand notes on page 11 and 12.

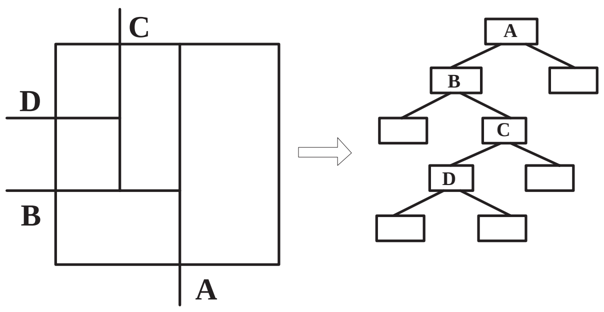


Figure 3: 2-D tree

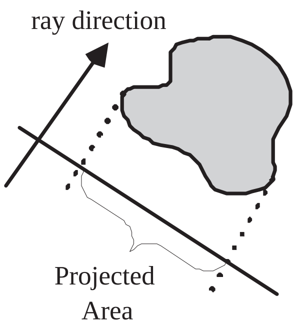


Figure 4: Projected surface area

3.2 Traversal Algorithm

Traversing the structure is recursive. As illustrated in Fig.5, t_{min}, t_{max} are near and far intersection points of the ray with the current volume; t^* is the intersection of ray with partitioning plane. In the left case, $t_{max} < t^*$, we recurse by

```
Intersect(Lvolume, tmin, tmax);
```

in the right case, $t_{min} < t^* < t_{max}$,, we recurse by

```
Intersect(Lvolume, tmin, t*);
Intersect(Rvolume, t*, tmax);
```

The above only applies for ray going to the right. The dot product of ray direction with plane normal $\vec{r} \cdot \vec{n}$ tells where the ray is going with respect to the plane.

Different structures can be used on different objects, depending on their own property and complexity. Always the acceleration structure is a type of the object.

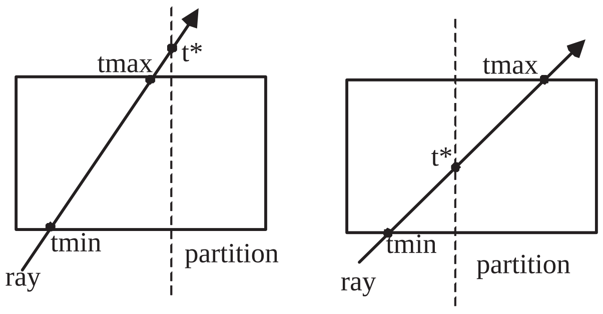


Figure 5: Recursive traversal algorithm

4 Hierarchical Bounding Volumes(HBV)

Trees of bounding volumes are constructed in a similar way to spatial hierarchy, however, HBV does not partition space: the bounding volumes can overlap each other. See Fig.6. HBV can be created from existing model hierarchy or by running a clustering algorithm.

The hierarchy itself can act as the whole model and be sent to rendering. Refer to *QSplat* paper from Stanford.

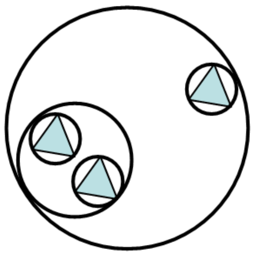


Figure 6: Hierarchical Bounding Volumes

5 Comparison

Q: Which one of the above is best?
A: I don't know.

Here all techniques are heuristics. Vlastimil Havran's Best Efficiency Scheme Project at `http://sgi.felk.cvut.cz/BES/` compares different accelerators. Basically every one of them works better in some cases and worse in some other cases. Worst case for "Uniform Grid" is sparse scenes, for example, a very complicated bunny model put at the center of a huge stadium.