Advanced Graphics Programming

Workshop Four – Acceleration Structures (Bounding Volume)

This workshop optimised the Ray Tracing algorithm using bounding volumes. Bounding volumes are the foundation for both Bounding Volume Hierarchies and Uniform Grid acceleration structures and standalone provide a significant performance increase to the ray tracing algorithm.

Step 1: Before optimising your ray tracing algorithm you should record some simple statistics such as:

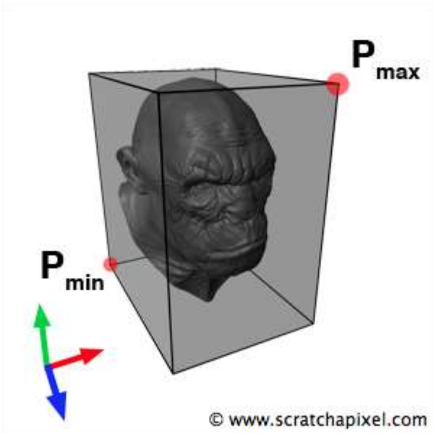
- The total time to render a frame (in seconds or milliseconds)
- Total number of rays cast in the scene
- Number of times the ray-primitive intersect function has been executed
- Number of ray-primitive hits

The render time can be captured with the chrono or the C-style date and time library. For millisecond accuracy use the std::chrono::high_resolution_clock::now function and make sure to include the chrono library. For measuring seconds use the clock function and include time.h. For more details: http://en.cppreference.com/w/cpp/chrono. The other statistics can be implemented as global variables.

You can output these statistics to the console window so that you can compare the efficiency of your algorithm before and after the optimisations.

***Note**: In Release mode the compiler will automatically optimise your code so always do your performance testing in Release mode instead of the default Debug mode.*

Step 2: First, add an axis aligned box class to your project for the bounding volume, it should have properties for $\mathbf{P}_{\mathbf{min}}$ and $\mathbf{P}_{\mathbf{max}}$ which represent the minimum and maximum extents of the box (see image below).



The axis aligned box class needs a ray intersection method which can be implemented by checking for intersecting with the 6 faces and returning the closest. An optimisation is to intersect only the front-facing planes. To get the distance to the intersection of each plane the following equations can be used, which should be repeated for the y and z components of the ray:

```
if rayDirection.x
       t1 = pMin.x – rayOrigin.x / rayDirection.x
       t2 = pMax.x – rayOrigin.x / rayDirection.x
```

To get the intersection point between the ray and a plane the following equation can be used, which can be repeated for the 6 planes:

```
       p1 = rayOrigin + rayDirection * t1
```

Finally, the intersection point must be checked to ensure it is on the face, which can be repeated for the 6 faces with `epsilon` set to `0.001f`:

```
       if p1.x > pMin.x – epsilon && p1.x < pMax.x + epsilon &&
          p1.y > pMin.y – epsilon && p1.y < pMax.y + epsilon &&
          p1.z > pMin.z – epsilon && p1.z < pMax.z + epsilon
              hit = true
```

If a face was hit then an intersection with the box has occurred otherwise there is no intersection.

Step 3: Calculate a bounding box for each object in the scene. For a triangle mesh imported as a model you can loop over all the vertices and find the minimum and maximum x, y and z co-ordinate and this will form the extent of the bounding box. The bounding volume of a sphere can be computed from its radius.

Warning: FLT_MIN returns the lowest possible positive float value. The actual lowest possible float value is the negative of FLT_MAX.

Step 4: Update your ray tracing algorithm to check first for ray intersections with the bounding volume of each object before checking ray intersections with the shapes within than bounding box. As illustrated by the following pseudocode:

```
for each object in scene
    if intersect(ray, object->boundingBox) == true)
        for (each shape in object) {
            if (intersect(ray, shape) == true) {
                // the ray intersects the shape
                ...
```

The following results show a significant performance increase using the bounding volume approach which can be further improved by extending the approach to Bounding Volume Hierarchies or the Uniform Grid acceleration structures.

	Before Bounding Volume	After Bounding Volume
Teapot	42 seconds	6 seconds
Teapot + hard shadows	64 seconds	11 seconds
Teapot + soft shadows	1,800 seconds	407 seconds