UMD CS Computer Graphics

Distribution Ray Tracing

Pete Willemsen

University of Minnesota Duluth

April 16, 2013

Outline

Accelerating the Ray Tracer

Objective

Reduce the overall complexity by focusing on being more efficient about which intersection tests to perform.

Efficiency by limiting the intersection tests

- What is the complexity of your ray tracers, thus far?

Accelerating the Ray Tracer

Objective

Reduce the overall complexity by focusing on being more efficient about which intersection tests to perform.

Efficiency by limiting the intersection tests

- What is the complexity of your ray tracers, thus far?
- Focus is on complexity of a single intersection test
- $O(N)$, in which a ray must be checked with N objects in the scene
- Essentially, linear search.

Reducing the complexity

- Need to bring complexity down to sub-linear time
- We'll start by find a simpler intersection test to perform

Bounding Boxes

Axis-aligned box that surrounds your objects. Uses the minimum and maximum extents in the three dimensions to form the box around an object. Can be specified with two 3D vectors.

Bounding Boxes

- Different than intersection tests for spheres and triangles
- Why?

Bounding Boxes

- Different than intersection tests for spheres and triangles
- Why?
- With spheres and triangles (or other objects), intersection test computes where the ray hits the object (*i.e.* $\vec{p}(t) = \vec{r}_o + t\vec{r}_d$)
- With bounding boxes, we only need to know **IF** the ray hits the box, not where

Bounding Box Intersection Tests

Start by examining the 2D version of the bounding box.

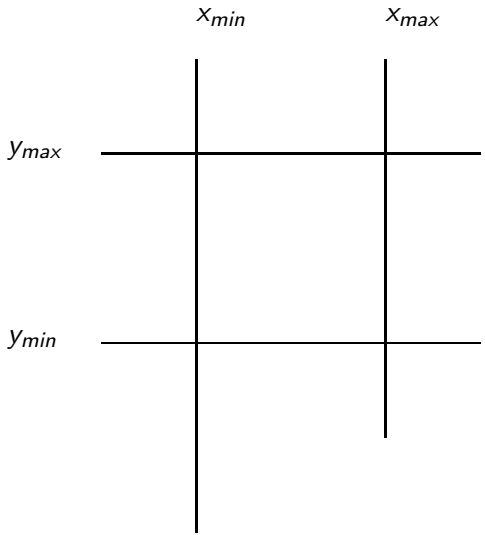
2D Bounding Box Definition

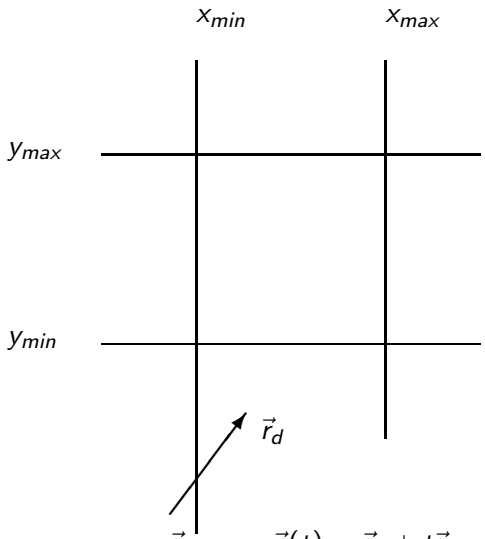
Bounding box is defined by the boundaries:

$$x_{min}, x_{max}, y_{min}, y_{max}$$

A point (x, y) is in the bounding box if

$$(x, y) \in [x_{min}, x_{max}] \times [y_{min}, y_{max}]$$


x_{min}
x_{max}
y_{max}
y_{min}


x_{min}
x_{max}
y_{max}
y_{min}
$\vec{r}_d$
$\vec{r}_o$
$\vec{p}(t) = \vec{r}_o + t\vec{r}_d$

Compute intersections of Bounding Box

Recall that our ray is defined by

$$\vec{p}(t) = \vec{r}_o + t\vec{r}_d$$

That's a vector equation, but we can also think of it in terms of the sub-components. For instance,

$$p_x = r_{o_x} + tr_{d_x}$$

$$p_y = r_{o_y} + tr_{d_y}$$

$$p_z = r_{o_z} + tr_{d_z}$$

Compute intersections of Bounding Box

If we substitute our bounding box boundary values for the components of the point of intersection, we can examine the sub-components for our bounding box intersection calculation. For instance,

$$x_{min} = r_{o_x} + tr_{d_x}$$

$$x_{max} = r_{o_x} + tr_{d_x}$$

$$y_{min} = r_{o_y} + tr_{d_y}$$

$$y_{max} = r_{o_y} + tr_{d_y}$$

And, for Z (but our examples in the slides will focus on looking at a 2D case with X and Y):

$$z_{min} = r_{o_z} + tr_{d_z}$$

$$z_{max} = r_{o_z} + tr_{d_z}$$

Compute intersections of Bounding Box

$$x_{min} = r_{o_x} + tr_{d_x}$$

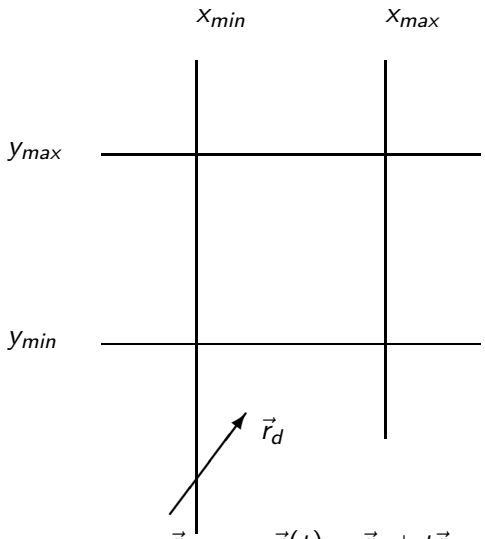
$$x_{max} = r_{o_x} + tr_{d_x}$$

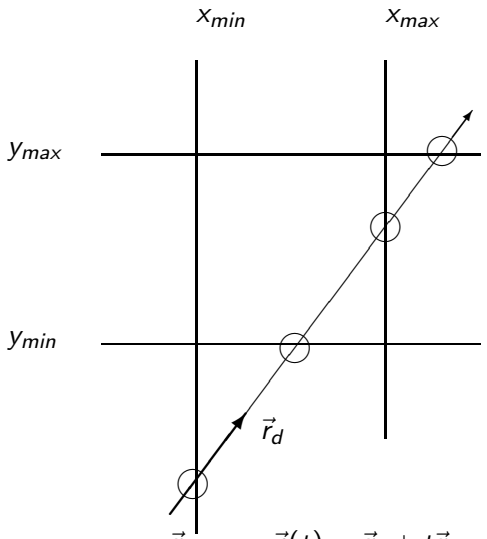
$$y_{min} = r_{o_y} + tr_{d_y}$$

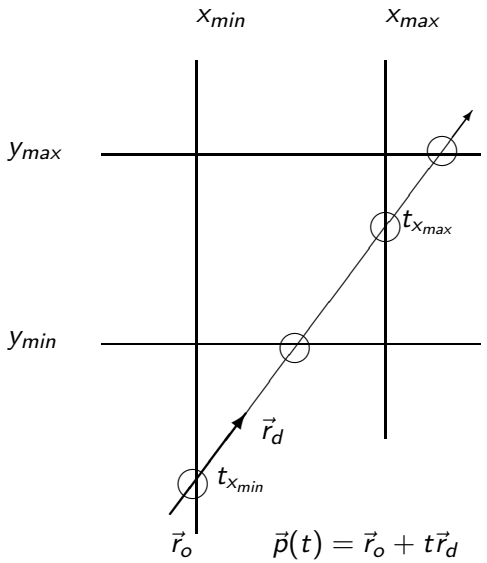
$$y_{max} = r_{o_y} + tr_{d_y}$$

So, solve for the t values associated with each of these parameters:

$$t_{x_{min}} = \frac{x_{min} - r_{o_x}}{r_{d_x}}$$


x_{min}
x_{max}
y_{max}
y_{min}
$\vec{r}_d$
$\vec{r}_o$
$\vec{p}(t) = \vec{r}_o + t\vec{r}_d$


x_{min}
x_{max}
y_{max}
y_{min}
$\vec{r}_d$
$\vec{r}_o$
$\vec{p}(t) = \vec{r}_o + t\vec{r}_d$


x_{min}
x_{max}
y_{max}
y_{min}
$t_{x_{max}}$
$\vec{r}_d$
$t_{x_{min}}$
$\vec{r}_o$
$\vec{p}(t) = \vec{r}_o + t\vec{r}_d$

Bounding Box Computation

$t_{x_{min}} = (x_{min} - r_{o_x})/r_{d_x}$
$t_{x_{max}} = (x_{max} - r_{o_x})/r_{d_x}$
$t_{y_{min}} = (y_{min} - r_{o_y})/r_{d_y}$
$t_{y_{max}} = (y_{max} - r_{o_y})/r_{d_y}$
if $(t_{x_{min}} > t_{y_{max}})$ or $(t_{y_{min}} > t_{x_{max}})$ **then**
  return false;
else
  return true;
end if

Bounding Box Computation

What happens when r_{d_x} or r_{d_y} is negative?

Bounding Box Computation

What happens when r_{d_x} or r_{d_y} is negative?

- The ray is basically coming from a different side
- Take into account when calculating the t values
- For instance, for x:

if $r_{d_x} \geq 0$ **then**
$t_{x_{min}} = (x_{min} - r_{o_x})/r_{d_x}$
$t_{x_{max}} = (x_{max} - r_{o_x})/r_{d_x}$
else
$t_{x_{min}} = (x_{max} - r_{o_x})/r_{d_x}$
$t_{x_{max}} = (x_{min} - r_{o_x})/r_{d_x}$
end if

You'll need to adapt for Y, and Z, of course.

Spatial Data Structures

Several mechanisms to accelerate ray intersection.

- Bounding Volume Hierarchy - bvh
- Uniform Spatial Subdivision
- Axis-Aligned Binary Space Paritioning
- many others...

Main goal: reduce unnecessary ray-object intersections!

Bounding Volume Hierarchy

Basic Idea Hierarchically surround objects by larger bounding boxes. For instance,

subdivide the classroom space into smaller components:

1. Start with a set of objects in the room, surround with one bounding box.
2. Break the set into two sets down the middle
3. Repeat Step 1 with each of the new sets unil each set is only one object

Use this tree structure in the intersection computations!

BVH Intersection - Roughly

```
if (ray hits largest box) then
  if (ray hits left subtree box) then
    check components of the left box for intersection
  if (ray hits right subtree box) then
    check components of the left box for intersection

  if anything was hit
    return correct intersection information
```

Rays could hit both subtrees! Why?

BVH Considerations

- Restrict tree to binary tree (others are possible - Quad or Oct)
- Each node *must* have a bounding box
- bvhNode is a type of Shape!

Why?

BVH Considerations

- Restrict tree to binary tree (others are possible - Quad or Oct)
- Each node *must* have a bounding box
- bvhNode is a type of Shape!

Why?

- Fits in perfectly with our general *intersection* interface!

BVH Node

```
class bvhNode : public RenderableObject {
    ...
  bool intersect(...);
  RenderableObject *leftChild;
  RenderableObject *rightChild;
    ...
};
```

Note that all RenderableObjects must have bounding boxes!

BVH Intersection

```
bool bvhNode::intersect(Ray r, t0, t1, hit)
```

BVH Intersection

```
bool bvhNode::intersect(Ray r, t0, t1, hit)

    if (bbox.hit(r, t0, t1)) then
```

BVH Intersection

```
bool bvhNode::intersect(Ray r, t0, t1, hit)

   if (bbox.hit(r, t0, t1)) then

      leftHit = leftChild->intersect(r, t0, t1, lhit)
      rightHit = rightChild->intersect(r, t0, t1, rhit)
```

BVH Intersection

```
bool bvhNode::intersect(Ray r, t0, t1, hit)

   if (bbox.hit(r, t0, t1)) then

      leftHit = leftChild->intersect(r, t0, t1, lhit)
      rightHit = rightChild->intersect(r, t0, t1, rhit)

      if (leftHit && rightHit) then
         // Inspect the data from the left and right
         // intersections, returning the closest!
      else if (leftHit)
         // only left side was hit
      else if (rightHit)
         // only right side was hit
```

Next up...

Next up, on Thursday:

- Understanding the BVH - bounding volume hierarchy
- Choices associated with creating the BVH
- Mesh objects

BVH Creation

What are ways we can create the BVH?

- Sort shapes along axis and split, using different axes at each level
- Parition the list about midpoint of an axis, using different axes at each level
- Actually, many other different ways, some better than others