CS 636
Advanced Rendering Techniques
Dr. David Breen
Online
Wednesday 6PM → 8:50PM
Presentation 3
4/22/20

Questions from Last Week?
- Color models
- Light models
- Phong shading model
- Assignment 2

Ray/Plane Intersection
Ray is defined by \( R(t) = R_0 + R_d \cdot t \), where \( t \geq 0 \)
\( R_0 \) = Origin of ray at \((x_0, y_0, z_0)\)
\( R_d \) = Direction of ray [\(x_d, y_d, z_d\)] (unit vector)

Plane is defined by \([A, B, C, D]\)
\( Ax + By + Cz + D = 0 \) for a point in the plane
Normal Vector, \( N = [A, B, C] \) (unit vector)
\( A^2 + B^2 + C^2 = 1 \)
\( D = -N \cdot P_0 \) (\(P_0\) - point in plane)

Ray/Plane (cont.)
Substitute the ray equation into the plane equation:
\( A(x_0 + x_d \cdot t) + B(y_0 + y_d \cdot t) + C(z_0 + z_d \cdot t) + D = 0 \)
Solve for \( t \):
\( t = \frac{-Ax_0 - By_0 - Cz_0 - D}{(Ax_d + By_d + Cz_d)} \)
\( t = -\frac{N \cdot R_0 - N \cdot P_0}{N \cdot R_d} \)
First check that \( N \cdot R_d \) not equal to zero!

Slide Credits
- Leonard McMillian, Seth Teller, Fredo Durand, Barb Cutler - MIT
- David Luebke - University of Virginia
- Matt Pharr - Stanford University
- Jonathan Cohen - Johns Hopkins U.
- Kevin Suffern - University of Technology, Sydney, Australia

More Geometry & Intersections
What Can Happen?

N • R

t < 0

t > 0

Ray/Plane Summary

Intersection point:

(x, y, z) = (x_o + x_d t, y_o + y_d t, z_o + z_d t)

1. Calculate N • R_d and compare it to zero.
2. Calculate t and compare it to zero.
3. Compute intersection point.
4. Flip normal if N • R_d is positive

Ray-Parallelepiped Intersection

- Axis-aligned
- From (X1, Y1, Z1) to (X2, Y2, Z2)
- Ray P(t)=R_o+R_d t

Naïve ray-box Intersection

- Use 6 plane equations
- Compute all 6 intersection
- Check that points are inside box
- The sign of Ax+By+Cz+D tells you if point is above or below the plane

Factoring out computation

- Pairs of planes have the same normal
- Normals have only one non-0 component
- Do computations one dimension at a time
- Maintain t_near and t_far (closest and farthest so far)

Test if parallel

- If R_o x = 0, then ray is parallel
  - if R_o x < X1 or R_o x > X2 return false
If not parallel
- Calculate intersection distance $t_1$ and $t_2$
  - $t_1 = (X_1 - R_{ox})/R_{dx}$
  - $t_2 = (X_2 - R_{ox})/R_{dx}$

Test 1
- Maintain $t_{near}$ and $t_{far}$
  - If $t_1 > t_2$, swap
  - If $t_1 > t_{near}$, $t_{near} = t_1$
  - If $t_2 < t_{far}$, $t_{far} = t_2$
- If $t_{near} > t_{far}$, box is missed

Test 2
- If $t_{far} < 0$, box is behind

Algorithm recap
- Do for all 3 axes
  - Calculate intersection distance $t_1$ and $t_2$
  - Maintain $t_{near}$ and $t_{far}$
  - If $t_{near} > t_{far}$, box is missed; Done
  - If $t_{far} < 0$, box is behind; Done
  - If box survived tests, return intersection at $t_{near}$
  - If $t_{near}$ is negative, return $t_{far}$

Motivation

Extra rays needed for these effects
- Distributed Ray Tracing
- Soft shadows
- Anti-aliasing (getting rid of jaggies)
- Glossy reflection
- Motion blur
- Depth of field (focus)
Shadows
- one shadow ray per intersection per point light source

Soft Shadows
- multiple shadow rays to sample area light source

Antialiasing – Supersampling
- multiple rays per pixel

Reflection
- one reflection ray per intersection

Glossy Reflection
- multiple reflection rays

Motion Blur
- Sample objects temporally
Depth of Field

- multiple rays per pixel

Algorithm Analysis

- Ray casting
- Lots of primitives
- Recursive
- Distributed Ray Tracing Effects
  - Soft shadows
  - Anti-aliasing
  - Glossy reflection
  - Motion blur
  - Depth of field

Algorithm Analysis (cont.)

- cost ≤ height * width * num primitives * intersection cost * num shadow rays * supersampling * num glossy rays * num temporal samples * max recursion depth * ... can we reduce this?

Bounding Regions

Goal: Reduce the number of ray/primitive intersection tests

Acceleration of Ray Casting

- Goal: Reduce the number of ray/primitive intersection tests

Conservative Bounding Region

- First check for an intersection with a conservative bounding region
- Early reject

Conservative Bounding Regions

- Tight → avoid false positives
- Fast to compute
- Fast to intersect

Bounding Volumes

- What makes a “good” bounding volume?
  - Tightness of fit (expressed how?)
  - Easy to compute
  - Simplicity of intersection
- Total cost = \(b \times B + i \times I + S\)
  - \(b\): # times volume tested for intersection
  - \(B\): cost of ray-volume intersection test
  - \(i\): # times item is tested for intersection
  - \(I\): cost of ray-item intersection test
  - \(S\): cost to compute BV parameters

Bounding Volumes

- Spheres
  - Cheap intersection test
  - Poor fit
  - Somewhat costly to fit to data

Bounding Volumes

- Axis-aligned bounding boxes (AABBs)
  - Relatively cheap intersection test
  - Usually better fit
  - Trivial to fit to data

Bounding Volume Diagram

Ray Tracing: Efficient Ray Tracing Techniques. Image: [Image URL]
Bounding Volumes
- Oriented bounding boxes (OBBs)
  - Medium-expensive intersection test
  - Very good fit (asymptotically better)
  - Medium-difficult to fit to data

Bounding Volumes
- Slabs (parallel planes)
  - Comparatively expensive
  - Very good fit
  - Very difficult to fit to data

Intersection with Axis-Aligned Box
- For all 3 axes, calculate the intersection distances $t_i$ and $t_j$.
- $t_{\text{near}} = \max (t_i, t_j, t_k)$
- $t_{\text{far}} = \min (t_i, t_j, t_k)$
- If $t_{\text{near}} > t_{\text{far}}$, box is missed.
- If $t_{\text{far}} < 0$, box is behind.
- If box survived tests, report intersection at $t_{\text{near}}$.

Bounding Box of a Triangle
- $(x_{\text{min}}, y_{\text{min}}, z_{\text{min}})$
- $(x_{\text{max}}, y_{\text{max}}, z_{\text{max}})$
- $(x, y, z) = (x_{\text{min}}, y_{\text{min}}, z_{\text{min}})$
- $(x, y, z) = (x_{\text{max}}, y_{\text{max}}, z_{\text{max}})$

Bounding Box of a Sphere
- $(x, y, z) = (x_{\text{center}}, y_{\text{center}}, z_{\text{center}})$
- $(x, y, z) = (x_{\text{center}} + r, y_{\text{center}} + r, z_{\text{center}} + r)$

Bounding Box of a Group
- $(x_{\text{min}_a}, y_{\text{min}_a}, z_{\text{min}_a})$
- $(x_{\text{min}_b}, y_{\text{min}_b}, z_{\text{min}_b})$
- $(x_{\text{max}_a}, y_{\text{max}_a}, z_{\text{max}_a})$
- $(x_{\text{max}_b}, y_{\text{max}_b}, z_{\text{max}_b})$
Spatial Data Structures

- Spatial partitioning techniques classify all space into non-overlapping portions
- Easier to generate automatically
- Can "walk" ray from partition to partition
- Hierarchical bounding volumes surround objects in the scene with (possibly overlapping) volumes
  - Often tightest fit

Spatial Partitioning

- Some spatial partitioning schemes:
  - Regular grid (2-D or 3-D)
  - Octree
  - k-D tree
  - BSP-tree

Acceleration Spatial Data Structures

Regular Grid

Create grid

- Find bounding box of scene
- Choose grid spacing
- grid, need not = grid,
Insert primitives into grid

- Primitives that overlap multiple cells?
- Insert into multiple cells (use pointers)

For each cell along a ray

- Does the cell contain an intersection?
- Yes: return closest intersection
- No: continue
- Use algorithm to step through cells

Preventing repeated computation

- Perform the computation once, "mark" the object
- Don’t re-intersect marked objects

Don’t return distant intersections

- If intersection t is not within the cell range, continue (there may be something closer)

Where do we start?

- Intersect ray with scene bounding box
- Ray origin may be inside the scene bounding box
- Cell (i, j)

Is there a pattern to cell crossings?

- Yes, the horizontal and vertical crossings have regular spacing
- \( \Delta t = \text{grid} \times \text{dir} \)
What's the next cell?

if \( t_{\text{next},v} < t_{\text{next},h} \)
\[
  i \leftarrow \text{sign}, \\
  t_{\text{next}} = t_{\text{next},v}, \\
  t_{\text{next},v} \leftarrow dt_v
\]
else
\[
  j \leftarrow \text{sign}, \\
  t_{\text{next}} = t_{\text{next},h}, \\
  t_{\text{next},h} \leftarrow dt_h
\]

if \( \text{dir}_x > 0 \) then \( \text{sign}_x = 1 \) else \( \text{sign}_x = -1 \)
if \( \text{dir}_y > 0 \) then \( \text{sign}_y = 1 \) else \( \text{sign}_y = -1 \)

What's the next cell?

- 3DDDA – Three Dimensional Digital Difference Analyzer
- 3D Bresenham Algorithm

Pseudo-code

create grid 
insert primitives into grid
for each ray \( r \)
  find initial cell \( c(i,j) \), \( t_{\text{min}} \), \( t_{\text{next},v} \) & \( t_{\text{next},h} \)
  compute \( dt_v \), \( dt_h \), \( \text{sign}_x \) and \( \text{sign}_y \)
  while \( c \neq \text{NULL} \)
    for each primitive \( p \) in \( c \)
      intersect \( r \) with \( p \)
      if intersection in range found
        return \( c = \text{find next cell} \)

Regular Grid Discussion

- Advantages?
  - easy to construct
  - easy to traverse

- Disadvantages?
  - may be only sparsely filled
  - geometry may still be clumped in a small number of cells

Acceleration Spatial Data Structures

Adaptive Grids

- Subdivide until each cell contains no more than \( n \) elements, or maximum depth \( d \) is reached

Adaptive Grids

- Nested Grids
- Octree (Quadtree)
Primitives in an Adaptive Grid
- Can live at intermediate levels, or be pushed to lowest level of grid.

Octree/(Quadtree)

Adaptive Grid Discussion
- Advantages?
  - Grid complexity matches geometric density
- Disadvantages?
  - More expensive to traverse (especially octree)

k-D Trees
- k-D tree pros:
  - Moderately simple to generate
  - More adaptive than octrees
- k-D tree cons:
  - Less efficient to trace rays across
  - Moderately complex data structure

BSP Trees
- BSP tree pros:
  - Extremely adaptive
  - Simple & elegant data structure
- BSP tree cons:
  - Very hard to create optimal BSP
  - Splitting planes can explode storage
  - Simple but slow to trace rays across

Acceleration Spatial Data Structures

Bounding Volume Hierarchy
- What makes a “good” bounding volume hierarchy?
  - Grouped objects (or volumes) should be near each other
  - Volume should be minimal
  - Sum of all volumes should be minimal
  - Top of the tree is most critical
  - Constructing the hierarchy should pay for itself!
Bounding Volume Hierarchy
- Find bounding box of objects
- Split objects into two groups
- Recurse

Where to split objects?
- At midpoint
- Sort, and put half of the objects on each side
- Use modeling hierarchy
**Data Structure Pseudo-code**

```plaintext
each_axis = 0;
Make_BVH(object_list, each_axis, ptr);
struct bbox = BoundingBox(object_list);
If # of objects < Threshold
  struct.obj_list = object_list
Else
  If (each_axis % 3) == 0 Sort object centroids in X
    ElseIf (each_axis % 3) == 1 Sort object centroids in Y
    Else
      Sort object centroids in Z
      Split sorted list into two halves
      Make_BVH(left_obj_list, each_axis++, lptr)
      Make_BVH(right_obj_list, each_axis++, rptr)
  struct.lptr = lptr; struct.rptr = rptr;
  ptr = &struct;
  Return
```

**Intersection with BVH**

- Check subvolume with closer intersection first

**Intersection with BVH**

- Don't return intersection immediately if the other subvolume may have a closer intersection

**Intersection Pseudo-code**

```plaintext
Does ray intersect box?
intersect_BVH(box, ray, xsect_pt, t)
If no more subboxes
  Intersect geometry and return nearest xsect_pt & t
If hits: return xsect_pt = Null,
  Sort t's
  Call subbox of nearest t subbox
  intersect_BVH(subbox, ray, xsect_pt, t)
  If hit subbox?
    If xsect_pt == Null || time < time
      Intersect_BVH(subbox2, ray, xsect_pt, t)
    Set nearest xsect_pt and t
  Return
```

**Bounding Volume Hierarchy Discussion**

- **Advantages**
  - easy to construct
  - easy to traverse
  - binary
- **Disadvantages**
  - may be difficult to choose a good split for a node
  - poor split may result in minimal spatial pruning
- **Hint**
  - Alternate sorting in X, Y & Z

**Transformation Hierarchy**

- Group & Transformation hierarchy may not be a good spatial hierarchy

- Group & Transformation hierarchy may not be a good spatial hierarchy
What’s the best method?
- What kind of scene are you rendering?
  - Teapot in a stadium vs. uniform distribution
  - Impact on surface tessellation on distribution
- Parameter values are critical

Shoot Fewer Rays
- Adaptive depth control
  - Naïve ray tracer: spawn 2 rays per intersection until max recursion limit
  - In practice, few surfaces are transparent or reflective
  - Stop shadow ray at first intersection between start and light source
  - Just shoot the rays you need
  - Determine contribution of ray
    - Don’t shoot rays w/ contribution near 0%

Adaptive sampling
- Shoot rays coarsely, interpolating their values across pixels
- Where adjacent rays differ greatly in value, sample more finely
- Stop when some maximum resolution is reached

Generalized Rays
- Beams, cones, pencils
- Area sampling, rather than point sampling
- Geometric computations are tricky (expensive?)
- Problems with reflection/refractions

Wrap Up
- Discuss next programming assignment
  - Add an acceleration technique
    - Adaptive grid
    - Bounding volume hierarchy
  - Supersample image
- Discuss status/problems/issues with this week’s programming assignment