CS 431/636
Advanced Rendering Techniques

Dr. David Breen
University Crossings 149
Tuesday 6PM → 8:50PM

Presentation 4
4/22/08
Questions from Last Week?

- Color models
- Light models
- Phong shading model
- Assignment 2
Slide Credits

- Leonard McMillan, 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
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

point light source

- no shadow rays
- one shadow ray
Soft Shadows

- multiple shadow rays to sample area light source

area light source

penumbra  umbra  penumbra

one shadow ray

lots of shadow rays
Antialiasing – Supersampling

- multiple rays per pixel
  - point light
  - area light
  - jaggies
  - w/ antialiasing
Reflection

- one reflection ray per intersection
Glossy Reflection

- multiple reflection rays

polished surface

Justin Legakis
Motion Blur

- Sample objects temporally

Rob Cook
Depth of Field

- multiple rays per pixel

film

focal length

Justin Legakis
Ray Tracing Acceleration Techniques

- Faster Intersections
  - Faster Ray-Object Intersections
    - Object bounding volumes
    - Efficient intersection routines

- Fewer Intersections
  - Bounding volume hierarchies
  - Frustum culling
  - Space subdivision
  - Directional techniques

- Fewer Rays
  - Adaptive tree depth
  - Statistical optimizations

- Generalized Rays
  - Beam tracing
  - Cone tracing
  - Pencil tracing

Algorithm Analysis

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

\[
\text{cost} \leq \text{height} \times \text{width} \times \text{num primitives} \times \text{intersection cost} \times \text{num shadow rays} \times \text{supersampling} \times \text{num glossy rays} \times \text{num temporal samples} \times \text{max recursion depth} \times \ldots
\]

Can we reduce this?
Bounding Regions
Acceleration of Ray Casting

- Goal: Reduce the number of ray/primitive intersection tests
First check for an intersection with a conservative bounding region

Early reject
Conservative Bounding Regions

- tight → avoid false positives
- fast to intersect

- bounding sphere
- axis-aligned bounding box
- non-aligned bounding box
- arbitrary convex region (bounding half-spaces)
Bounding Volumes

What makes a “good” bounding volume?

- Tightness of fit (*expressed how?*)
- Simplicity of intersection

\[ \text{Total cost} = b \times B + i \times I \]

- \(b\): # 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
Bounding box speeds up rendering
Ray through this pixel tests the bounding box and intersects the object.

Bounding box slows down rendering
Bounding Volumes

- Spheres
  - Cheap intersection test
  - Poor fit
  - Somewhat expensive to fit to data
Bounding Volumes

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

arbitrary convex region (bounding half-spaces)
Intersection with Axis-Aligned Box

From Lecture 2

- For all 3 axes, calculate the intersection distances $t_1$ and $t_2$
- $t_{near} = \max (t_{1x}, t_{1y}, t_{1z})$
- $t_{far} = \min (t_{2x}, t_{2y}, t_{2z})$
- If $t_{near} > t_{far}$, box is missed
- If $t_{far} < 0$, box is behind
- If box survived tests, report intersection at $t_{near}$
Bounding Box of a Triangle

\[(x_{\text{min}}, y_{\text{min}}, z_{\text{min}}) = (\min(x_0, x_1, x_2), \min(y_0, y_1, y_2), \min(z_0, z_1, z_2))\]

\[(x_{\text{max}}, y_{\text{max}}, z_{\text{max}}) = (\max(x_0, x_1, x_2), \max(y_0, y_1, y_2), \max(z_0, z_1, z_2))\]
Bounding Box of a Sphere

\[
(x, y, z) = (x_{\text{min}}, y_{\text{min}}, z_{\text{min}}) = (x-r, y-r, z-r)
\]

\[
(x_{\text{max}}, y_{\text{max}}, z_{\text{max}}) = (x+r, y+r, z+r)
\]
Bounding Box of a Group

\[(x_{\text{min}_a}, y_{\text{min}_a}, z_{\text{min}_a}) = (\min(x_{\text{min}_a}, x_{\text{min}_b}), \min(y_{\text{min}_a}, y_{\text{min}_b}), \min(z_{\text{min}_a}, z_{\text{min}_b}))\]

\[(x_{\text{max}_a}, y_{\text{max}_a}, z_{\text{max}_a}) = (\max(x_{\text{max}_a}, x_{\text{max}_b}), \max(y_{\text{max}_a}, y_{\text{max}_b}), \max(z_{\text{max}_a}, z_{\text{max}_b}))\]
Acceleration Spatial Data Structures
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
Regular Grid
Create grid

- Find bounding box of scene
- Choose grid spacing
- $grid_x$ need not = $grid_y$
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
Preventing repeated computation

- Perform the computation once, "mark" the object
- Don't re-intersect marked objects
Don't return distant intersections

- If intersection 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
Is there a pattern to cell crossings?

Yes, the horizontal and vertical crossings have regular spacing.

\[
\begin{align*}
\text{dir}_x & \quad \text{dir}_y \\
\text{grid}_x & \quad \text{grid}_y \\
\end{align*}
\]

\[
\begin{align*}
dt_y &= \text{grid}_y / \text{dir}_y \\
dt_h &= \text{grid}_x / \text{dir}_x
\end{align*}
\]
What's the next cell?

\[
\text{if } t_{next_v} < t_{next_h} \\
i += \text{sign}_x \\
t_{min} = t_{next_v} \\
t_{next_v} += dt_v \\
\text{else} \\
j += \text{sign}_y \\
t_{min} = t_{next_h} \\
t_{next_h} += dt_h
\]

\[
\text{if } (\text{dir}_x > 0) \text{ sign}_x = 1 \text{ else } \text{sign}_x = -1 \\
\text{if } (\text{dir}_y > 0) \text{ sign}_y = 1 \text{ else } \text{sign}_y = -1
\]
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
Acceleration Spatial Data Structures

Adaptive Grids
Adaptive Grids

- Subdivide until each cell contains no more than $n$ elements, or maximum depth $d$ is reached
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
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
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse
Where to split objects?

- At midpoint  \( OR \)
- Sort, and put half of the objects on each side  \( OR \)
- Use modeling hierarchy
sort_in_x = TRUE;
Make_BVH(object_list, sort_in_x, ptr)
    struct.bbox = BoundingBox(object_list);
    If # of objects < Threshold
        struct.obj_list = object_list
    Else
        If (sort_in_x) Sort object centroids in X
        Else Sort object centroids in Y
        Split sorted list into two halves
        Make_BVH(left_obj_list, !sort_in_x, lptr)
        Make_BVH(right_obj_list, !sort_in_x, 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

Does ray intersect box?
intersect_BVH(box, ray, xsect_pt, t)
    If no more subboxes
    Intersect geometry and return nearest xsect_pt & t
    Intersect ray with both subboxes
    No hits: return xsect_pt = Null;
    Sort t’s
    Call subbox of nearest t subbox1
    intersect_BVH(subbox1, ray, xsect_pt, t)
    If hit_subbox2?
        If xsect_pt == Null || t_{2near} <= t_{1far}
            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 and Y
Group & Transformation hierarchy may not be a good spatial hierarchy

Transform B

Transform C

Transform D

Transform E

Flatten
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%
Shoot Fewer Rays

- 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
    - Uniform grid
    - Adaptive grid
    - Bounding volume hierarchy
  - Supersample image
- Discuss status/problems/issues with this week’s programming assignment