Acceleration Data Structures for Ray Tracing

Most slides are taken from Fredo Durand
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

point light

area light

jaggies

w/ antialiasing
Reflection

- one reflection ray per intersection
Glossy Reflection

- multiple reflection rays
Motion Blur

• Sample objects temporally
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?
The cost of Ray Tracing

- Many Primitives
- Many Rays
- Expensive Intersections
Reduce the number of ray/primitive intersections
Bounding Volumes

- Idea: associate with each object a simple bounding volume. If a ray misses the bounding volume, it also misses the object contained therein.
- Effective for additional applications:
  - Clipping acceleration
  - Collision detection
Early reject

- First check for an intersection with a conservative bounding region
Conservative Bounding Regions

- **Axis-aligned bounding box**
- **Non-aligned bounding box**
- **Bounding sphere**
- **Arbitrary convex region (bounding half-spaces)**

Diagram showing examples of each type of bounding region.
What is a good bounding volume?

• tight → avoid false positives
• fast to intersect
• easy to construct
Bounding Volumes
Bounding Volumes

(a)  

(b)  

(c)  

(d)
Hierarchical Bounding Boxes
Intersection with Axis-Aligned Box

• For all 3 axes, calculate the intersection distances $t_1$ and $t_2$
  
  $t_{\text{near}} = \max (t_{1x}, t_{1y}, t_{1z})$
  $t_{\text{far}} = \min (t_{2x}, t_{2y}, t_{2z})$

• If $t_{\text{near}} > t_{\text{far}}$, box is missed

• If $t_{\text{far}} < t_{\text{min}}$, box is behind

• If box survived tests, report intersection at $t_{\text{near}}$
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 volumes can intersect

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

- At midpoint \( OR \)
- Sort, and put half of the objects on each side \( OR \)
- Use modeling hierarchy
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
Spatial Subdivision
Spatial Subdivision

• Uniform spatial subdivision:
  – The space containing the scene is subdivided into a uniform grid of cubes “voxels”.
  – Each voxel stores a list of all objects at least partially contained in it.
  – Given a ray, voxels are traversed using a 3D variant of the 2D line drawing algorithms.
  – At each voxel the ray is tested for intersection with the primitives stored therein
  – Once an intersection has been found, there is no need to continue to other voxels.
Create grid

- Find bounding box of scene
- Choose grid spacing
- $\text{grid}_x$ need not = $\text{grid}_y$
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 $t$ is not within the cell range, continue (there may be something closer)
Is there a pattern to cell crossings?

- Yes, the horizontal and vertical crossings have regular spacing.

\[ dt_v = \frac{\text{grid}_y}{\text{dir}_y} \]
\[ dt_h = \frac{\text{grid}_x}{\text{dir}_x} \]
Where do we start?

- Intersect ray with scene bounding box
- Ray origin may be inside the scene bounding box
What's the next cell?

```plaintext
if \( t_{next_v} < t_{next_h} \)
    \( i += sign_x \)
    \( t_{min} = t_{next_v} \)
    \( t_{next_v} += dt_v \)
else
    \( j += sign_y \)
    \( t_{min} = t_{next_h} \)
    \( t_{next_h} += dt_h \)
```

If \( (dir_x > 0) \) \( sign_x = 1 \) else \( sign_x = -1 \)

If \( (dir_y > 0) \) \( sign_y = 1 \) else \( sign_y = -1 \)
What's the next cell?

- 3DDDA – Three Dimensional Digital Difference Analyzer

- We'll see this again later, for line rasterization
Uniform vs. Adaptive Subdivision
Regular Grid Discussion

• Advantages?
  – easy to construct
  – easy to traverse

• Disadvantages?
  – may be only sparsely filled
  – geometry may still be clumped
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)
Step from cell to cell. Intersect current cell and add an epsilon into the next cell. Then search for the cell in the tree. A naïve search starts from the root. Otherwise, try an intelligent guess…
Top down traversal

Split ray into sub-segments and traverse each segment recursively.
Kd-trees vs. Quad-tree
Kd-trees vs. BSP-tree
Adaptive Spatial Subdivision

• Disadvantages of uniform subdivision:
  – requires a lot of space
  – traversal of empty regions of space can be slow
  – not suitable for “teapot in a stadium” scenes

• Solution: use a hierarchical adaptive spatial subdivision data structure
  – octrees
  – BSP-trees

• Given a ray, perform a depth-first traversal of the tree. Again, can stop once an intersection has been found.
Uniform vs. Adaptive Subdivision
Proximity Clouds
Parallel/Distributed RT

• Two main approaches:
  – Each processor is in charge of tracing a subset of the rays. Requires a shared memory architecture, replication of the scene database, or transmission of objects between processors on demand.
  – Each processor is in charge of a subset of the scene (either in terms of space, or in terms of objects). Requires processors to transmit rays among themselves.
Directional Techniques

- Light buffer: accelerates shadow rays.
  - Discretize the space of directions around each light source using the *direction cube*
  - In each cell of the cube store a sorted list of objects visible from the light source through that cell
  - Given a shadow ray locate the appropriate cell of the direction cube and test the ray with the objects on its list
Directional Techniques

- Ray classification (Arvo and Kirk 87):
  - Rays in 3D have 5 degrees of freedom: \((x, y, z, \theta, \phi)\)
  - Rays coherence: rays belonging to the same small 5D neighborhood are likely to intersect the same set of objects.
  - Partition the 5D space of rays into a collection of 5D hypercubes, each containing a list of objects.
  - Given a ray, find the smallest containing 5D hypercube, and test the ray against the objects on the list.
  - For efficiency, the hypercubes are arranged in a hierarchy: a 5D analog of the 3D octree. This data structure is constructed in a lazy fashion.