Ray Tracing II: Acceleration Techniques
Overview

- Various ray-object intersection details
- Ray-object intersection a substantial computational cost
  - 50-90% of run time, depending on shading complexity
- Spatial subdivision, bounding volume hierarchies
Ray-object details

• **Object transformations**
  • Transform the ray origin and direction by the inverse transform
  • Transform the object if possible, though

• **Normalize ray direction vector?**
  • Can make intersection tests faster, but renormalizing after transform is slow
  • Comparing ray t values easier if not re-normalized after transform!
Shape Intersection Interface

- Intersect(): general rays
- IntersectP(): shadow rays: no geom. info
- WorldBound(): world space bounding box
- ObjectBound(): object space bbox
- CanIntersect(): can we call Intersect()?
- Refine(): new shapes
Local Differential Geometry

• Shape-independent method for representing intersection information
  • Point P
  • Normal N
  • Parametric \((u,v)\)
  • Partial derivatives
  • (Tangents, change in normal, ...)

Matt Pharr, Spring 2003
Ray Intersection Acceleration

- Problem: naive algorithm scales linearly with scene complexity
- Solution: don’t use the naive algorithm!
- Four main options
  - Faster ray-object intersections
  - Fewer ray-object intersections
  - Fewer rays
  - Generalized rays
Faster Ray-Object Intersections

- Micro-optimization techniques
- SSE/4 rays at once via SIMD (Wald et al)
- maxt to quickly cull objects
- Shadow rays don’t need differential geometry
Tracing Generalized Rays

- Beams, cones, pencils, ...
- Area sampling rather than point sampling
- Geometric computations are tricky
- Problems with refraction, ...

Matt Pharr, Spring 2003
Fewer Ray-Object Intersections

- Shadow rays are special: any intersection will do
  - Stop after first hit
  - Shadow cache
  - Light buffer
  - Shaft culling
- Backface culling
- Bounding volumes
Basic Bounding Volumes

- Surround object with a simple volume
- Test ray against volume first
- Cost model: \( n \times c_b + p_i \times n \times c_o \)
- \( n \) is given; minimize \( c_b \) and \( p_i \)
- Spheres, boxes...
- Test object-space or world-space bound?
Spatial Data Structures

• 3+D data structure
  • Two main approaches
    • Spatial Subdivision
    • Bounding Volume Hierarchies
  • Does the hierarchy drive the subdivision of space, or do the bounds of the objects drive it?
Uniform Grids: Creation

- Find bounding box
- Choose # of voxels: $k \times \sqrt[3]{n}$
- Engrid objects
  - Use bounds to find candidate voxels
  - Possibly do voxel-object overlap test
Uniform Grids: Traversal

- Intersect ray with grid bounds
- Use DDA to step through voxels
  - Like Bresenham, but must visit all voxels the ray passes through!
- Compute intersections with objects in each voxel
Objects that overlap multiple voxels

- Continue until intersection in current voxel
- Early out for shadow rays, though
- Mailboxes to eliminate redundant intersection tests
- Assign rays numbers, check against objects last-tested-ray number
- Not so good for multi-threading...
Hierarchical Grids

- Solves the lack-of-adaptivity problem
- Can re-use DDA setup computations
- Effective in practice
Hierarchical Spatial Subdivision

- Recursive subdivision of space
  - Octree, kd-tree, bsp-tree
  - 1-1 Relationship between points in the scene and leaf nodes of the tree
- Example: point location by recursive search (log time)
Creating Spatial Hierarchies

- Top down versus bottom up
- Top down:

```c
Create(node, prims) {
    if (# prims < MAX_PRIMS ||
        depth > MAX_DEPTH)
        add(prims, node->prims);
    else {
        refine(node);
        foreach node->child
            Create(child, overlap(prims, child));
    }
}
```
Traversing Spatial Hierarchies

- Recursive traversal from top node
- Maintain front-to-back todo list
- Examples...
Other Approaches

- Bounding volume hierarchies
  - Kay-Kajiya: heap based on t distance to bounding volume intersection
- 5D ray hierarchies
- Meta-hierarchies
So What’s Best?

- Every method has been conclusively proven to be better than all of the others.
- SPD scenes popular, though dated
- V. Havran, Best Efficiency Scheme Project
Really, What’s Best?

- What kinds of scenes do you want to render?
  - “Teapot in a stadium” versus uniform distribution
  - Impact of tessellation of patches/subdivision surfaces on distribution?
- Constant factors are critically important
- Adaptivity generally key for robustness
- Cache effects becoming more important
Asymptotic Running Time

• Triangles (Pellegrini)
  • Time: \( O(\log n) \)
  • Space: \( O(n^{5+\epsilon}) \)

• Spheres (Guibas and Pellegrini)
  • Time: \( O(\log^2 n) \)
  • Space: \( O(n^{5+\epsilon}) \)

• In practice, log-ish behavior generally seen