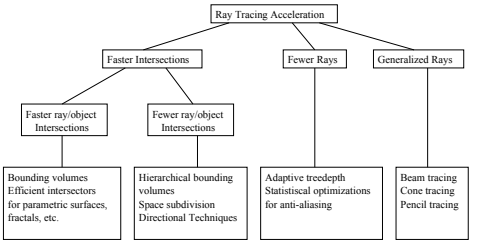
Raytracing: Performance

COSC 4328/5327
Scott A. King

Ray Genealogy

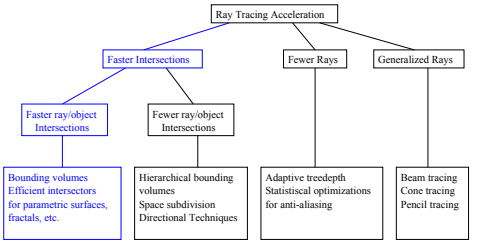
- 1 Ray/pixel at 1k × 1k image = 1M rays.
- Say on avg. 6 secondary rays = 7M rays
- 100k objects with 10 ops for intersection = ? Operations 7, 000, 000M ops
- 4GHz processor 5 cycles per op = 800MFLOPS
- How long to render? ~2.5hr
- **Where are we spending the time?**
- How can we improve performance?
- Take advantage of Coherence
  Image coherence – close pixels display same object
  Spatial coherence – close points similarly colored
  Temporal coherence – pixels change little each frame

Acceleration Classification



James Arvo and David Kirk

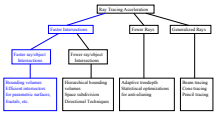
Acceleration Classification



James Arvo and David Kirk

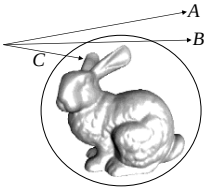
Bounding Volumes

- Enclose objects/primitives inside volume with simpler intersection test.
- For objects that are intersected do we have an increase or decrease in computation?
- For objects away from ray do we have an increase or decrease in calculations?
- Which case happens more often?



Bounding Volume

- Ray-bunny intersection takes 70K ray-triangle intersections even if ray misses the bunny
- Place a sphere around bunny
  - Ray *A* misses sphere so ray *A* misses bunny without checking 70K ray-triangle intersections
  - Ray *B* intersects sphere but still misses bunny after checking 70K intersections
  - Ray *C* intersects sphere and intersects bunny
- Can also use axis-aligned bounding box
  - Easier to create for triangle mesh



Bounding Volumes

- Spheres
- Boxes (parallelepipeds)
- Slabs (pairs of parallel planes)

$$Cost = n * B + m * I$$

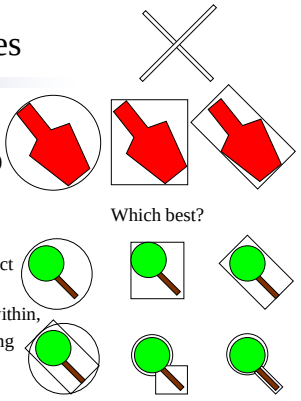
where

n - number of rays,

m - number of rays that intersect bounding volume,

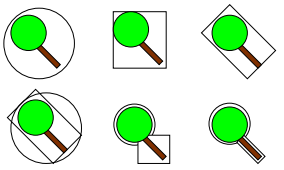
I - cost of intersecting object within,

B - cost of intersecting bounding volume.



Bounding Volumes Performance

- Tradeoff complexity versus closeness of fit.
- Transformed bounding volumes.
- Intersection of bounding volumes.
- Union of bounding volumes.
- What about hierarchies?

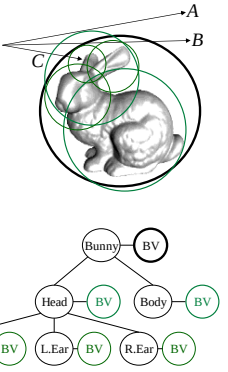


Why Hierarchies

- What is the complexity of the number of intersection tests for bounding volumes?
- What if you create a tree-like structure of bounding volumes?
- So using hierarchies can give a theoretical logarithmic time complexity.
- When enclosing several volumes with a new volume, the cost of doing the extra check must pay off.
- Hierarchies are not always simple to construct.

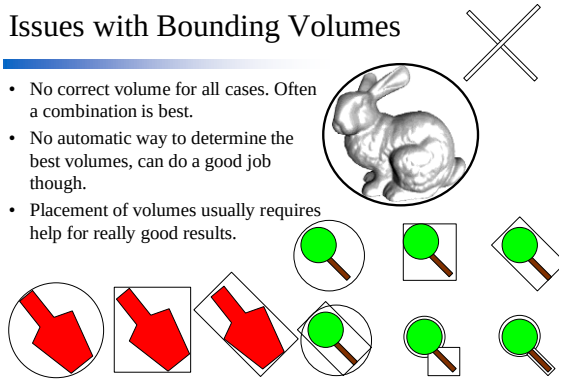
Bounding Volume Hierarchy

- Associate bounding volume with each node of scene graph
- If ray misses a node's bounding volume, then no need to check any node beneath it
- If ray hits a node's BV, then replace it with its children's BV's (or geometry)
- Breadth first search of tree
  - Maintain heap ordered by ray-BV intersection t-values
  - Explore children of node w/least pos. ray-BV t-value



Issues with Bounding Volumes

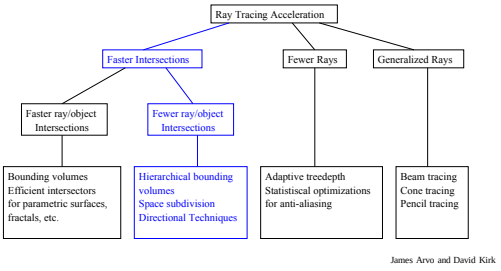
- No correct volume for all cases. Often a combination is best.
- No automatic way to determine the best volumes, can do a good job though.
- Placement of volumes usually requires help for really good results.



Spatial Subdivision

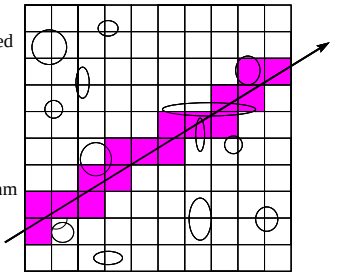
- Bounding volumes divide the space based on the objects.
- Instead lets just divide the space.
- Divide the space into *voxels*.

Acceleration Classification



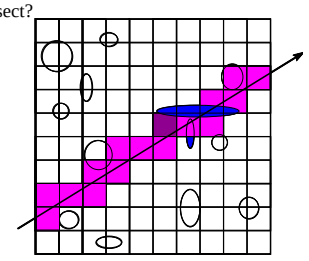
Uniform Space Subdivision

- Divide space into equal size blocks (voxels)
- Test only voxels intersected by rays.
  - Notice anything interesting?
- How do we determine the next voxel to test?
  - 3D Bresenham algorithm



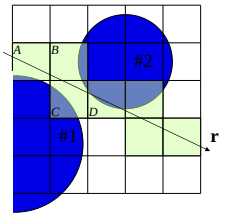
Issues

- Which object does this ray intersect?
- Consider this object?
  - Which voxels does it intersect?
- What happens at this voxel?
  - mailbox



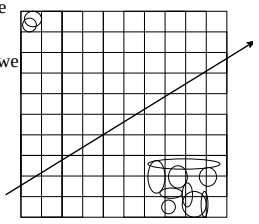
Tagging

- Ray-object intersection test valid for ray with entire object
  - not just portion of object inside current cell
- Need only intersect object once for each ray
- In cell *A* – list = {#1}
  - Intersect **r** with #1? Yes
    - Miss → Tag #1 with no-intersection
- In cell B – list = {#2}
  - Intersect **r** with #2? Yes
    - ray **r** hits object #2 but later in cell *C*
    - Tag object #2 with intersection-at-*C*
- In cell *C* – list = {#1,#2}
  - Intersect **r** with #1? No (no-intersection)
  - Intersect **r** with #2? No (intersection-at-*D*)
- In cell *D* – list = {#2}
  - Intersect **r** with #2? No (intersection-at-*D*)



Issues with uniform

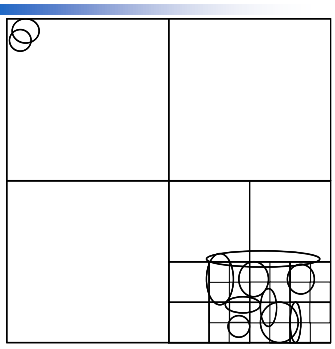
- What affect does the size of the voxels have?
  - More voxels →
    - Increased traversal time
    - Tighter fit of bounding volume
    - More memory
- Ideally, how many voxels/object do we want?
- Is this good or bad?
  - Why?



Nonuniform (hierarchical)

- Instead of having lots of empty little cells, lets have just a few empty big cells.
- This gives us a tree structure (hierarchy again!)
- Less voxels.
- But at what cost?
- Octrees.
- BSP trees.

Quadtrees

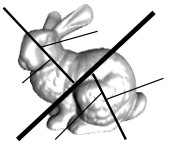


Octrees

- Divide until a cell reaches minimum # of objects (often 1) or min voxel size.
- Can subdivide on the fly (dynamic) to help improve efficiency of octree.
  - Divide if voxel large or many objects
  - Divide if more than N rays (4 is good) and at least one hit object.
  - Divide if MK < N
    - M - # rays through cell that hit
    - k (2 or higher) user defined weight
  - If a voxel working why subdivide!

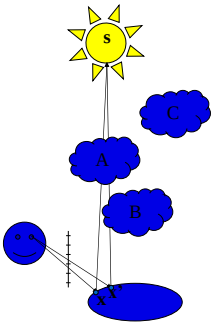
Other Partitioning Structures

- Octree
  - Ray can parse through large empty areas
  - Requires less space than grid
  - Subdivision takes time
- Binary Space Partition (BSP) Tree
  - Planes can divide models nearly in half
  - Trees better balanced, shallower
  - Added ray-plane intersections



Shadow Caching

- Any interloper between surface point **x** and the light source **s** will cast a shadow
  - Doesn't matter how many
  - Doesn't matter which is closest
  - Stop ray intersections once *any* intersection found
- Neighboring shadowed surface points **x** and **x**' probably shadowed by the same object
  - Start shadow ray intersection search with object intersected in last shadow search



Other ways to increase

- $O(cNM)$
  - N – Number of Rays
  - M – Number of Objects
- $O(cNlog(M))$
- What about c
  - Don't do extra work, only normalize if you have to (only do once)
  - Only calc hit point, Normal, etc. if it is the object hit (smallest +t)
- Parallel – ridiculously parallel
- Threads – best if bundle

Nonuniform vs Uniform

Extensions to Specification