Acceleration Data Structures for Ray Tracing

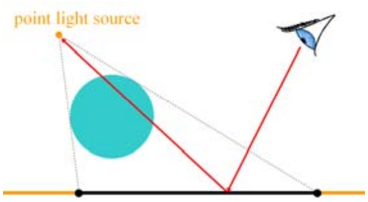
Most slides are taken from Fredo Durand

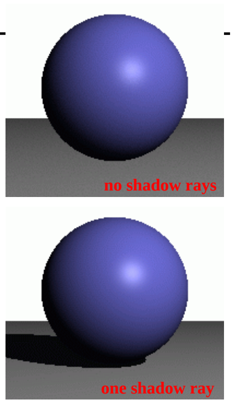
Extra rays needed for these effects:

- Distribution Ray Tracing
  - Soft shadows
  - Anti-aliasing (getting rid of jaggies)
  - Glossy reflection
  - Motion blur
  - Depth of field (focus)

Shadows

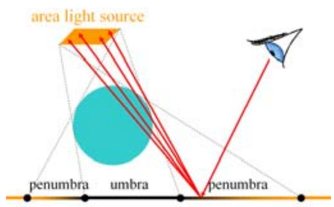
- one shadow ray per intersection per point light source

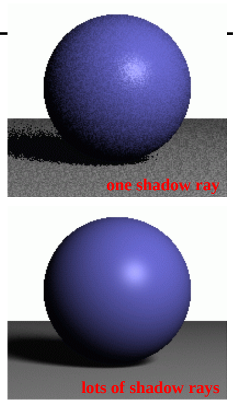




Soft Shadows

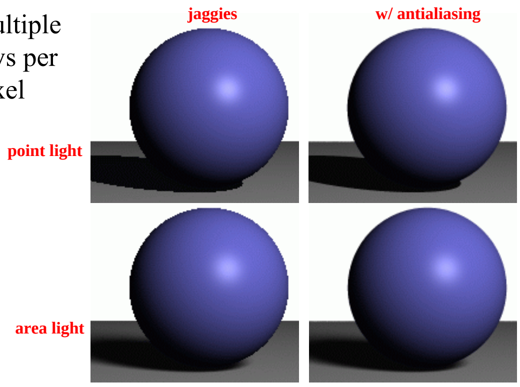
- multiple shadow rays to sample area light source





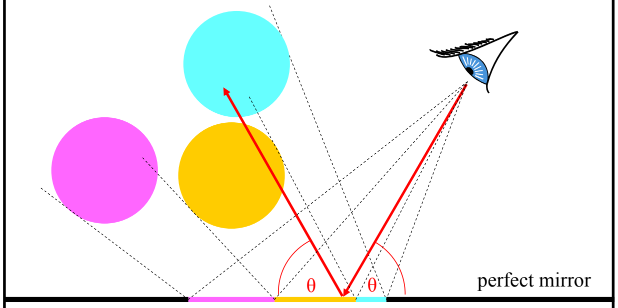
Antialiasing – Supersampling

- multiple rays per pixel



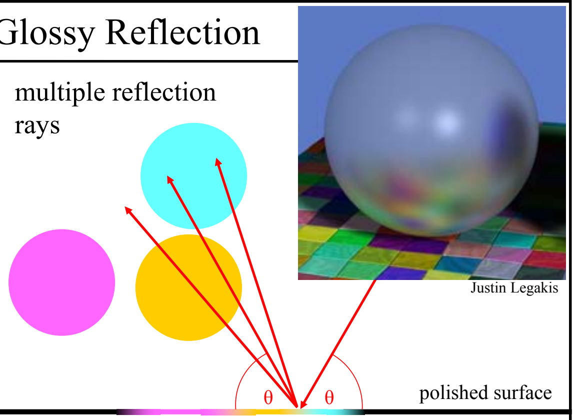
Reflection

- one reflection ray per intersection



Glossy Reflection

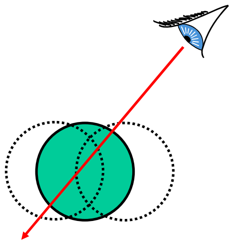
- multiple reflection rays



Justin Legakis

Motion Blur

- Sample objects temporally



Rob Cook

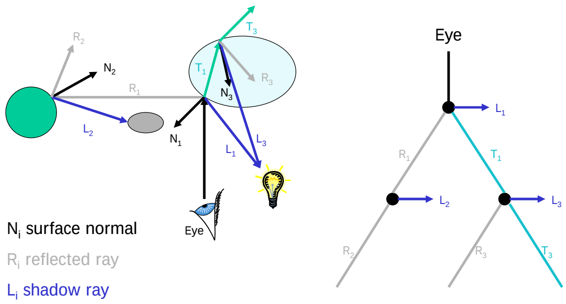
Algorithm Analysis

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

cost ≤ height * width *
num primitives *
intersection cost *
num shadow rays *
supersampling *
num glossy rays *
num temporal samples *
max recursion depth *
. . .

can we reduce this?

The Ray Tree



N_i surface normal

R_i reflected ray

L_i shadow ray

T_i transmitted (refracted) ray

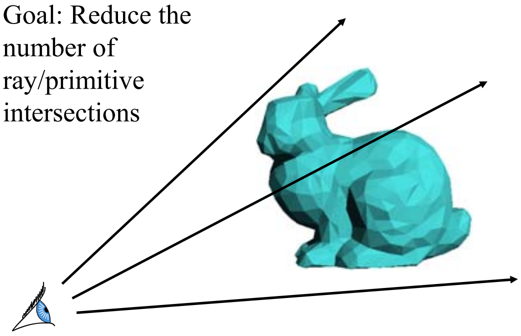
Questions?

Accelerating Ray Tracing

- Four main groups of acceleration techniques:
  - Reducing the average cost of intersecting a ray with a scene:
    - Faster intersection calculations
    - Fewer intersection calculations
  - Reducing the total number of rays that are traced
    - Adaptive recursion depth control
  - Discrete Ray Tracing
    - proximity clouds
  - Using generalized rays
  - Parallelization, specialized hardware

Acceleration of Ray Casting

- Goal: Reduce the number of ray/primitive intersections

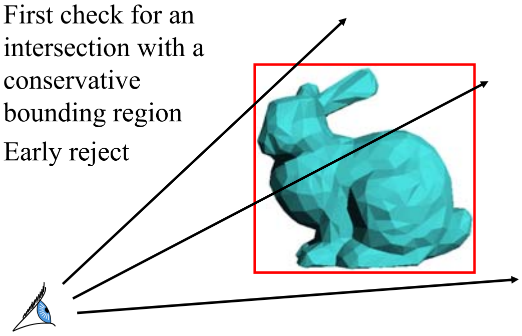


Bounding Volumes

- Idea: associate with each object a simple bounding volume. If a ray misses the bounding volume, it also misses the object contained therein.
- Common bounding volumes:
  - spheres
  - bounding boxes
  - bounding slabs
- Effective for additional applications:
  - Clipping acceleration
  - Collision detection
- Note: bounding volumes offer no asymptotic improvement!

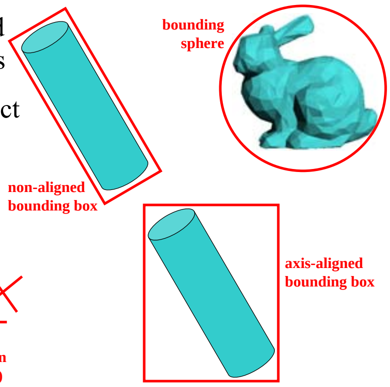
Conservative Bounding Region

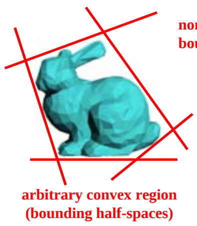
- First check for an intersection with a conservative bounding region
- Early reject



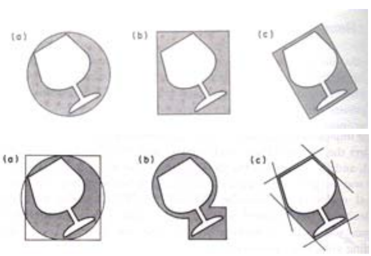
Conservative Bounding Regions

- tight → avoid false positives
- fast to intersect

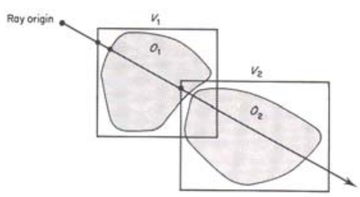




Bounding Volumes

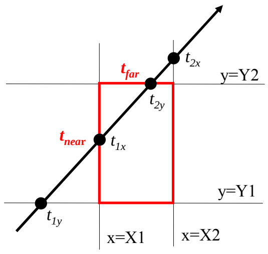


Bounding Boxes can overlap



Intersection with Axis-Aligned Box

From Lecture 3,
Ray Casting II



- 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 $t_{far} < t_{min}$, box is behind
- If box survived tests, report intersection at t_{near}

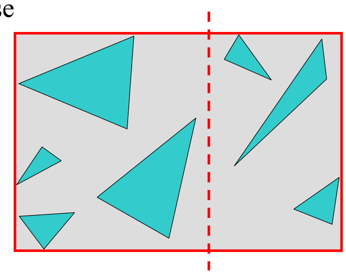
Bounding Volume Hierarchy

- Introduced by James Clark (SGI, Netscape) in 1976 for efficient view-frustum culling.

```
Procedure IntersectBVH(ray, node)
begin
   if IsLeaf(node) then
      Intersect(ray, node.object)
   else if IntersectBV(ray,node.boundingVolume)
   then
      foreach child of node do
         IntersectBVH(ray, child)
      endfor
   endif
end
```

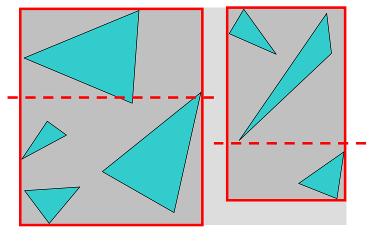
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



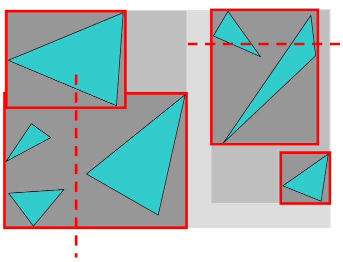
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



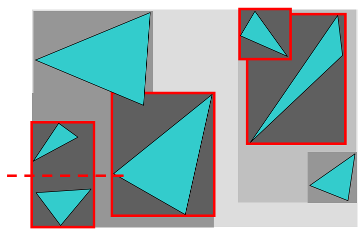
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



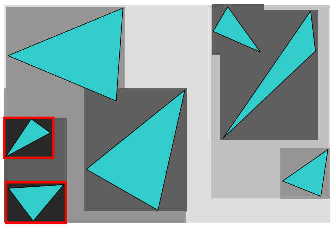
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



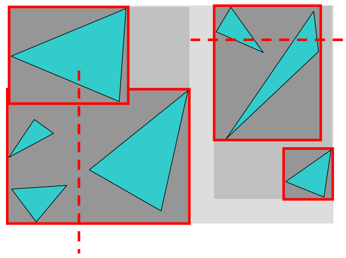
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



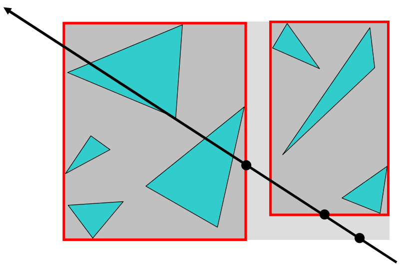
Where to split objects?

- At midpoint *OR*
- Sort, and put half of the objects on each side *OR*
- Use modeling hierarchy



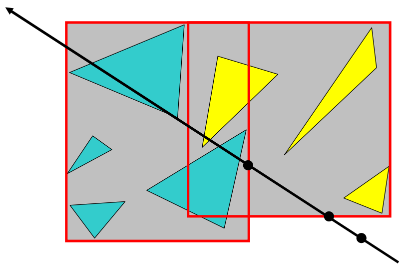
Intersection with BVH

- Check subvolume with closer intersection first



Intersection with BVH

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

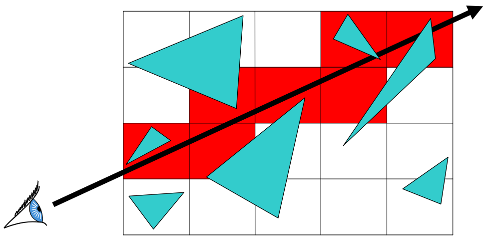


Questions?

Spatial Subdivision

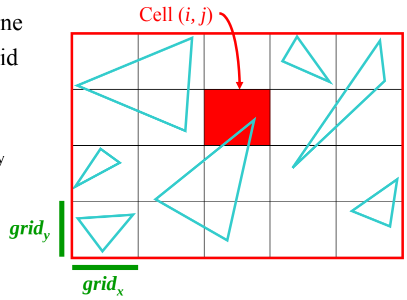
- Uniform spatial subdivision:
  - The space containing the scene is subdivided into a uniform grid of cubes “voxels”.
  - Each voxel stores a list of all objects at least partially contained in it.in
  - Given a ray, voxels are traversed using a 3D variant of the 2D line drawing algorithms.
  - At each 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.

Regular Grid



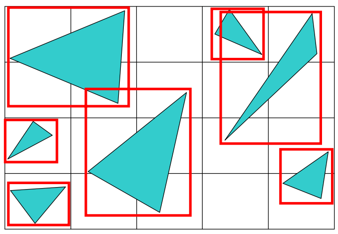
Create grid

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



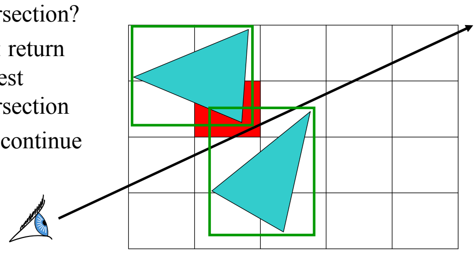
Insert primitives into grid

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



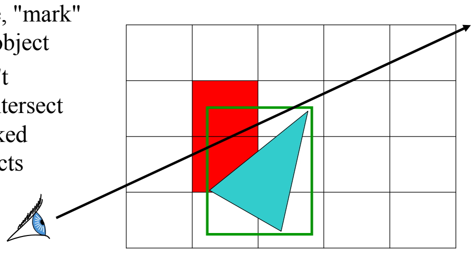
For each cell along a ray

- Does the cell contain an intersection?
- Yes: return closest intersection
- No: continue



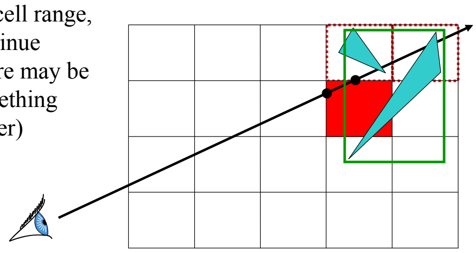
Preventing repeated computation

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



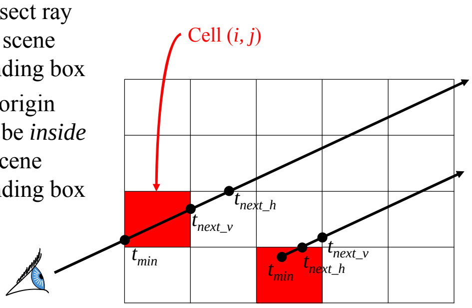
Don't return distant intersections

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



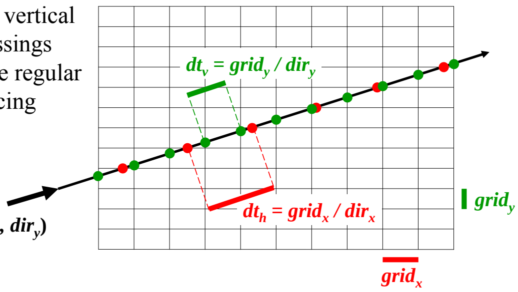
Where do we start?

- Intersect ray with scene bounding box
- Ray origin may be *inside* the scene bounding box



Is there a pattern to cell crossings?

- Yes, the horizontal and vertical crossings have regular spacing



What's the next cell?

if $t_{next\_v} < t_{next\_h}$

i += $sign_x$

$t_{min} = t_{next\_v}$

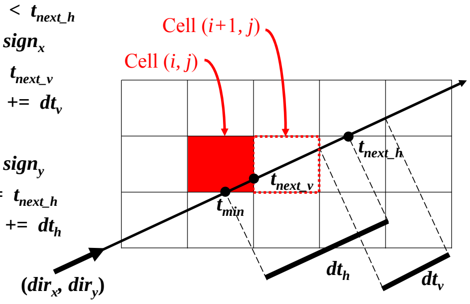
t_{next_v} += dt_v

else

j += $sign_y$

$t_{min} = t_{next\_h}$

t_{next_h} += dt_h

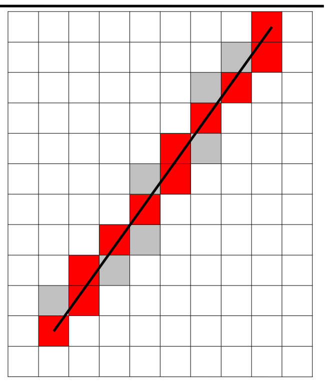


if $(dir_x > 0)$ $sign_x = 1$ *else* $sign_x = -1$

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



Pseudo-code

```
create grid
insert primitives into grid
for each ray r
  find initial cell c(i,j), t_min, t_next_v & t_next_h
  compute dt_v, dt_h, sign_x and sign_y
  while c != NULL
    for each primitive p in c
      intersect r with p
      if intersection in range found
        return
    c = find next cell
```

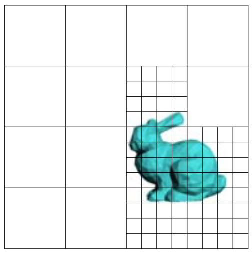
Regular Grid Discussion

- Advantages?
  - easy to construct
  - easy to traverse
- Disadvantages?
  - may be only sparsely filled
  - geometry may still be clumped

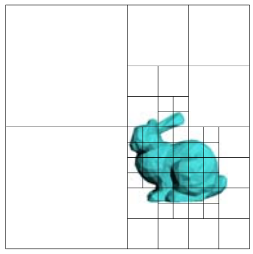
Questions?

Adaptive Grids

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



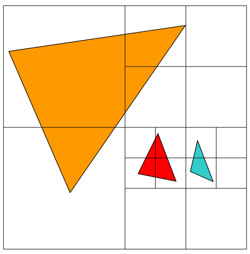
Nested Grids



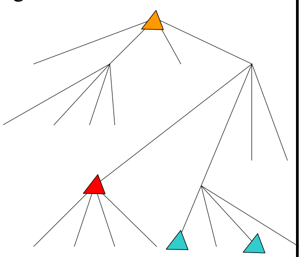
Octree/(Quadtree)

Primitives in an Adaptive Grid

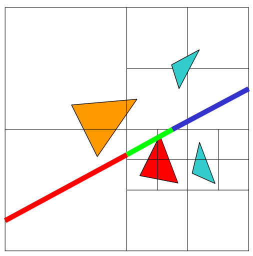
- Can live at intermediate levels, or be pushed to lowest level of grid



Octree/(Quadtree)

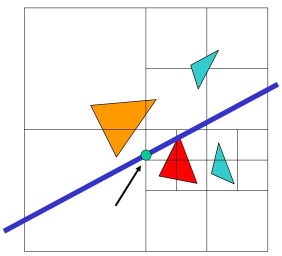


Top down traversal



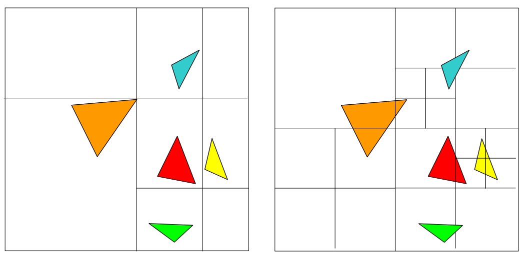
Split ray into sub-segments and traverse each segment recursively.

Bottom Up traversal

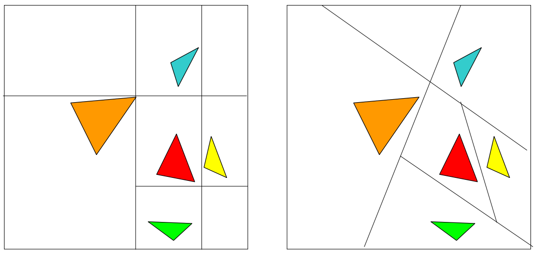


Step from cell to cell.
Intersect current cell and add an epsilon into the next cell.
Then search for the cell in the tree.
A naïve search starts from the root.
Otherwise, try an intelligent guess…

Kd-trees vs. Quad-tree



Kd-trees vs. BSP-tree



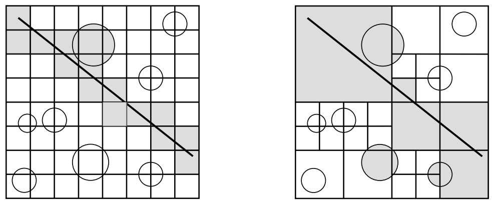
Adaptive Spatial Subdivision

- 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.

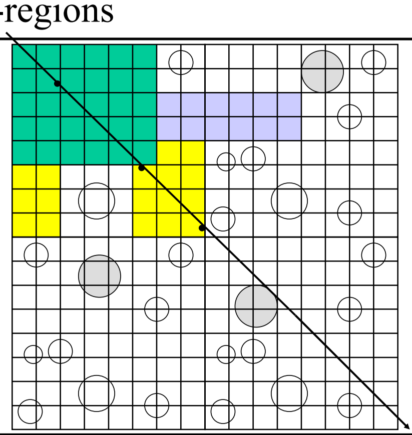
Bounding Volume Hierarchy Discussion

- 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

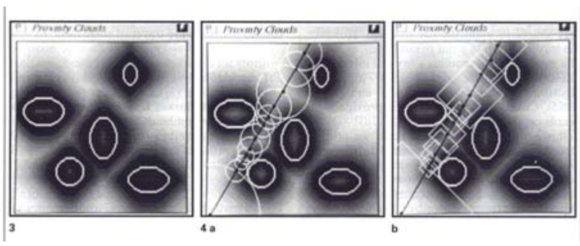
Uniform vs. Adaptive Subdivision

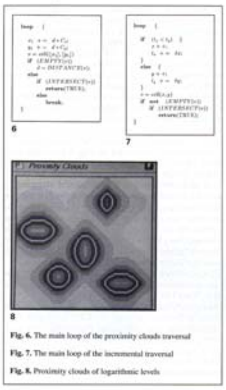


Macro-regions



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,θ,ϕ)
  - 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.