Computer graphics
Ray tracing

Putting it all together

Our last real lecture!!

Computer Graphics Rendering
- World is represented by a set of 3D objects, with colors, reflectivity, transparency, etc.
  - Primitive objects: Polygons, spheres, cones
  - Complex objects: Mesh of triangles
- Goal: Produce a realistic 2D picture of the world

Ray-tracing algorithm
Input: - world: set of 3D objects
       - (x,y,z) position of the eye
       - Position of the 2D screen
Output: Image: array of colors of size nPixels by mPixels
For i = 1…nPixels
  For j = 1…mPixels
    r = ray( eye -> pixel(i,j) )
    object = getClosestIntersection(r, world)
    if (object!=null) then
      image[I,j] = object.getColor();
Finding intersections

- Suppose your world consists of Millions of objects
- How can you calculate closest intersection quickly?
  - Computing intersection between ray and each object is much too slow
- Idea: Store your objects in a data structure that allows you to quickly discard objects that can’t have intersection

Quad trees

For a 2D-world, Subdivide the world into four quadrants.
Keep subdividing as long there is more than one object per square
For 3D-world, Subdivide world into eight octants

Fast ray intersection problem

To quickly find intersection between ray and world:
Find which main quadrant is intersected
Find which of its subquadrant is intersected
… Keep going down the tree until a leaf is found
If leaf contains an object, test intersection
Continue until intersection is found