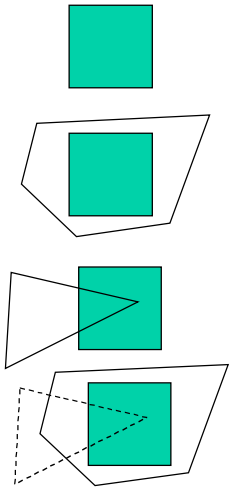
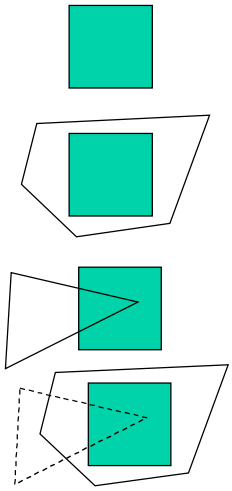
Area subdivision

- Four tractable cases for a given region in image plane:
  - no surfaces project to the region
  - only one surface completely surrounds the region
  - only one surface is completely inside the region or overlaps the region
  - a polygon is completely in front of everything else in that region determined by considering depths of the polygons at the corners of the region



Area subdivision

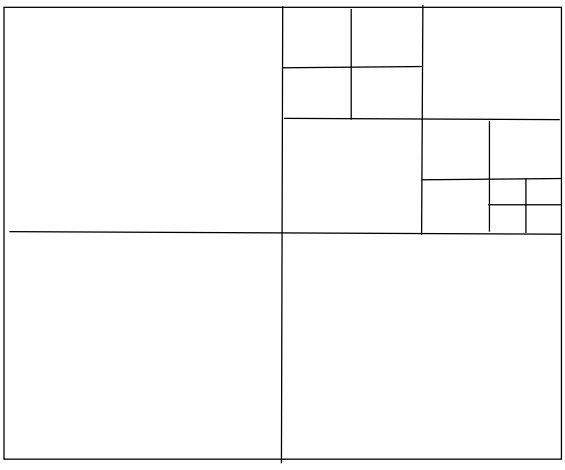
- Four tractable cases for a given region in image plane:
  - no surfaces project to the region (do nothing or paint background)
  - only one surface completely surrounds the region (paint surface)
  - only one surface is completely inside the region or overlaps the region (do nothing or paint background and then scan convert region)
  - a polygon is completely in front of everything else in that region determined by considering depths of the polygons at the corners of the region (paint surface)



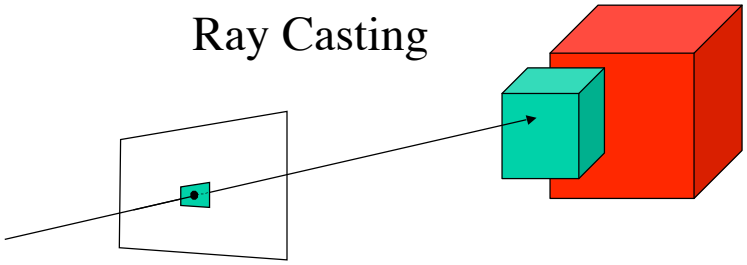
Area subdivision

- Algorithm:
  - subdivide each region until one of these cases is true or until region is very small
  - if case is true, deal with it
  - if region is small, choose surface with smallest depth.
  - determining cases quickly makes use of the same ideas in depth sort (depth sorting, bounding boxes, tests for which side of the plane), and the difficult cases are deferred by further subdivision.
- Advantages:
  - can be very efficient
  - no over rendering
  - anti-aliases well (subdivide a bit further)

One Subdivision Strategy



Ray Casting



- Image precision algorithm
- For each pixel cast a ray into the world
  - For each surface
    - determine intersection point with ray
  - Render pixel based on closest surface

Ray Casting

- First step in ray tracing algorithm
- Expensive
- Good performance usually requires clever data structures such as bounding volumes for object groups or storing world occupancy information in octrees.
- Other main problem is computing intersection.
- For polygons, we can use the standardized orthographic space where we can work in 2D.
- Spheres are easy (a bit more difficult in perspective space).
- Useful for “picking”--not expensive here (why?)

Ray Tracing--teaser

- Idea is very simple--follow light around
- Following **all** the light around is intractable, so we follow the light that makes **the most** difference
- Work backwards from what is seen
- Simple ray tracer
  - Cast a ray through each pixel (as in ray casting for visibility)
  - From intersection point cast additional rays to determine the color of the pixel.
    - For diffuse component, must cast rays to the lights
    - We may also add in some "ambient" light
    - For mirrors, must cast ray in mirror direction (recursion--what is the stopping condition)

Current state of intro students graphic's ability

Know how to draw polygons

Know about cameras

Know how to map 3D polygons onto the screen

Know how to draw the bits closest to the cameras

Issues

Should we live in a polygonal world?

How do you get polygons for complex objects?

What color should each pixel be?

Coloring pixels

Need to model light and surface

Simplest model

Point light source and Lambertian (diffuse) reflection. Gives basic shader--makes things look 3D

Point light source

Modeled by single light direction (key attribute, more than "point-like"--e.g., the sun is essentially a point source)