Visibility - So Far

Back face culling

(Did everyone understand the E-mail?)

Painters algorithm

Z and A buffer (briefly, will review)

The Z - buffer

- For each pixel on screen, have a second memory location - called the z-buffer
- Set this buffer to a value corresponding to the furthest point
- As a polygon is filled in, compute the depth value of each pixel
  - if depth < z buffer depth, fill in pixel and new depth
  - else disregard
- Typical implementation: Compute Z while scan-converting. A ∂Z for every ∂X is easy to work out.

The Z - buffer

- Advantages:
  - simple; hardware implementation common
  - efficient z computations are easy.
- Disadvantages:
  - over renders - can be slow for very large collections of polygons - may end up scan converting many hidden objects
  - quantization errors can be annoying (not enough bits in the buffer)
  - doesn’t do transparency, filtering for anti-aliasing.

The A - buffer

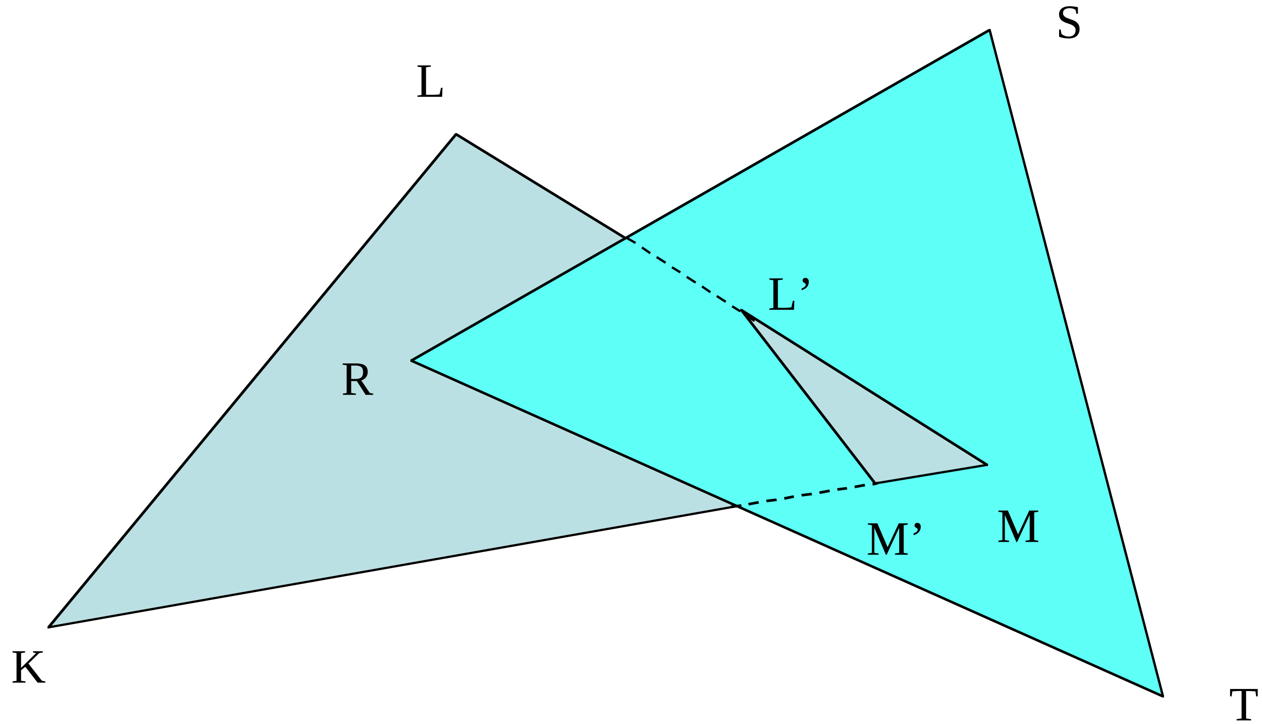
- For transparent surfaces and filter anti-aliasing:
- Algorithm: filling buffer
  - at each pixel, maintain a pointer to a list of polygons sorted by depth.
  - When filling a pixel:
    - if polygon is opaque and covers pixel, insert into list, removing all polygons farther away
    - if polygon is opaque and only partially covers pixel, insert into list, but don’t remove farther polygons
- Algorithm: rendering pixels
  - at each pixel, traverse buffer using brightness values in polygons to fill.
  - values are used either in transparency or for filtering
- Adv:
  - can do more than z-buffer
- Disadv:
  - over renders
  - quantization errors can be annoying

Scan line algorithm

- Assume polygons do not intersect one another.
- Observation: on any given scan line, the visible polygon can change only at an edge.
- Algorithm:
  - fill all polygons simultaneously (details are in §13.3)
  - at each scan line, have all edges that cross scan line in AEL
  - keep record of current depth at current pixel - use to decide which is in front in filling span

Scan line algorithm

- To deal with penetrating polygons, split them up



Scan line algorithm

- Advantages:
  - potentially fewer quantization errors (more bits available for depth)
  - filter anti-aliasing can be made to work.
- Disadvantages:
  - invisible polygons clog AEL, ET (can easily be more expensive than Z-buffer over-rendering).

Depth sorting

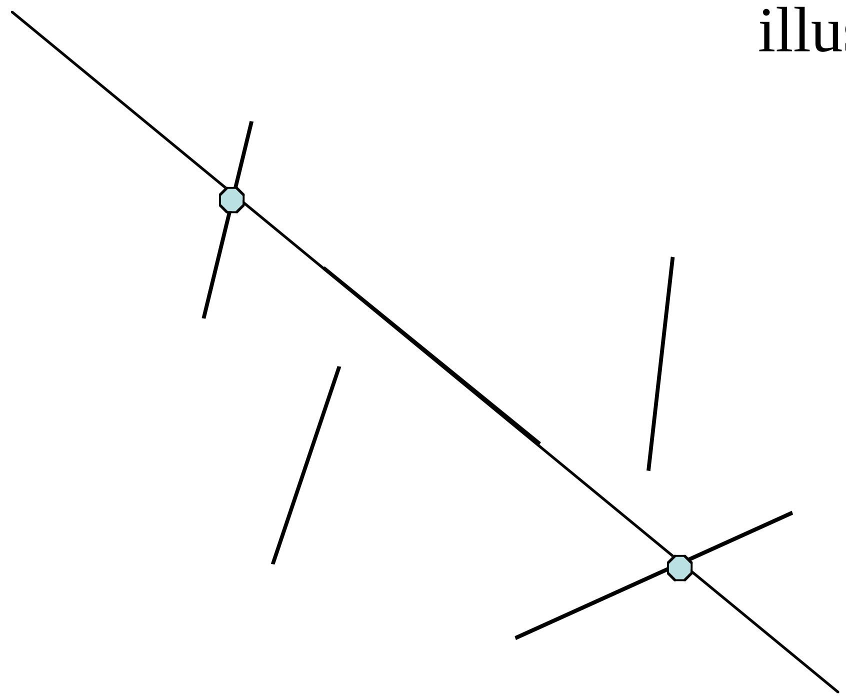
- Sort in order of decreasing depth
- Render in sorted order
- Rendering:
  - for surface S with greatest depth
    - if no depth overlaps, render (like painter's alogirthm)
    - if depth overlaps, test for problem overlap in image plane
    - if S, S' overlap in depth and in image plane, swap and try again
    - if S, S' have been swapped already, split and reinsert
- Testing image plane problem overlaps (test get incresinly expensive):
  - xy bounding boxes do not intersect
  - *or* S is behind the plane of S'
  - *or* S' is in front of the plane of S
  - *or* S and S' do not intersect
- Advantages:
  - filter anti-aliasing works fine
  - no depth quantization error
- Disadvantages:
  - over rendering

BSP - trees

- Construct a tree that gives a rendering order
- Tree splits 3D world into cells, each of which contain at most one piece of polygon.
- Constructing tree:
  - Choose polygon (arbitrary)
  - split its cell using plane on which polygon lies
  - continue until each cell contains only one polygon

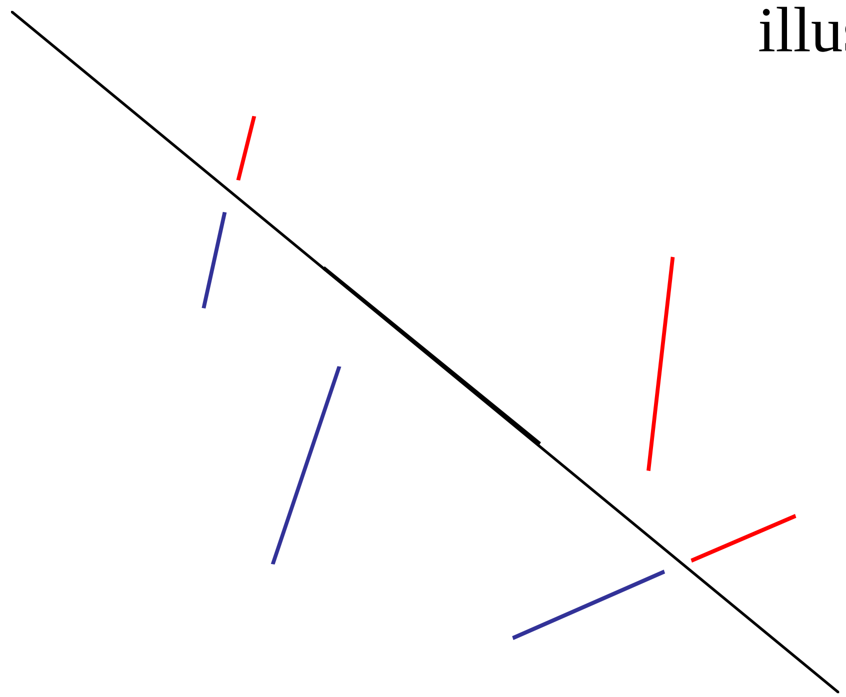
BSP - trees

2D version for illustration



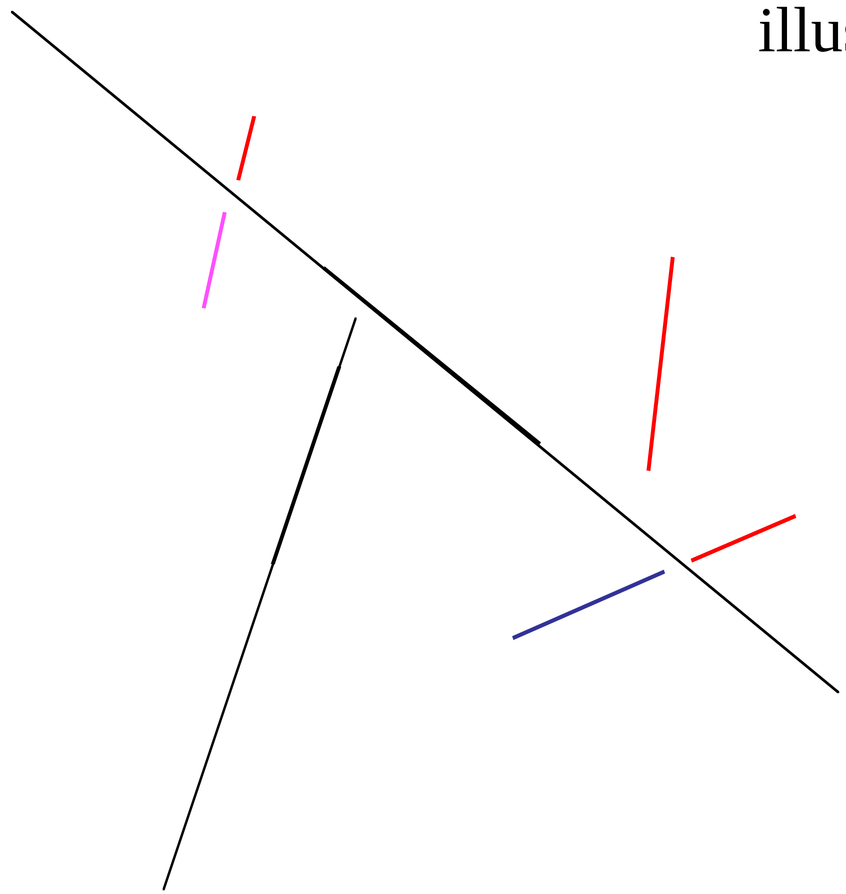
BSP - trees

2D version for illustration



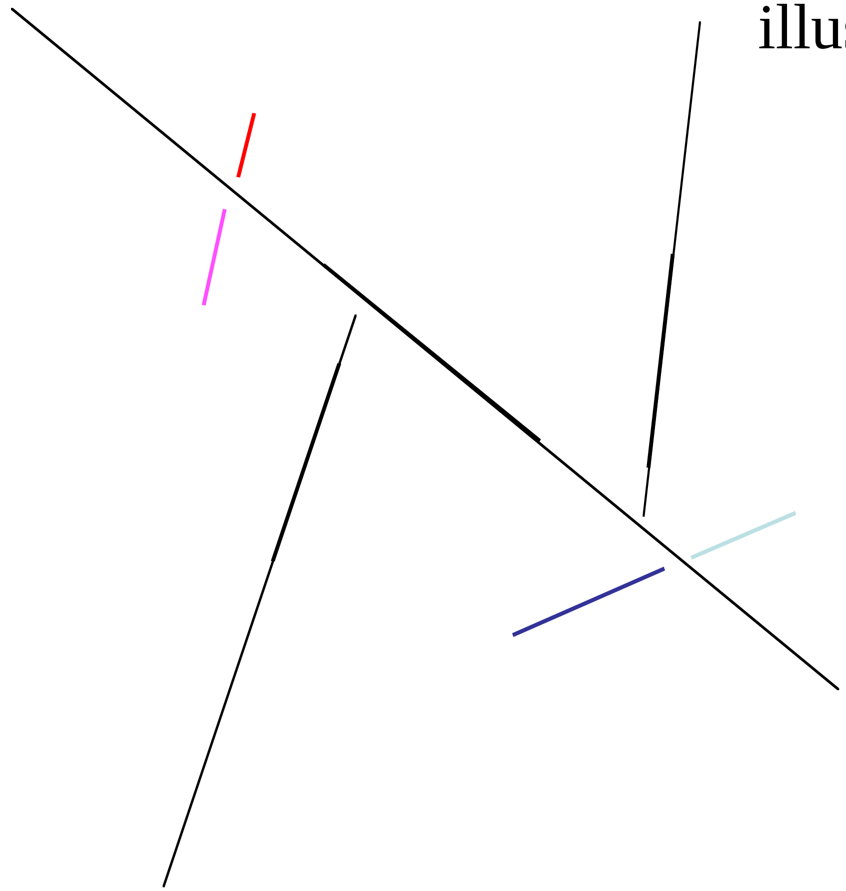
BSP - trees

2D version for illustration



BSP - trees

2D version for illustration

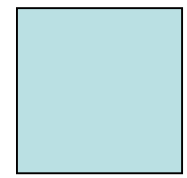


BSP - trees

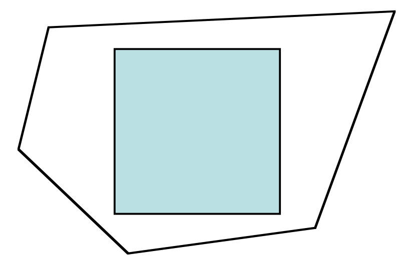
- Rendering tree:
  - recursive descent
  - render back, node polygon, front
- Disadvantages:
  - many small pieces of polygon (more splits than depth sort!)
  - over rendering
  - hard to get balanced tree
- Advantages:
  - one tree works for any focal point (good for cases when scene is static)
  - filter anti-aliasing works fine, as does transparency

Area subdivision

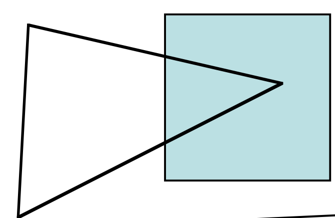
- Four tractable cases for a given region in image plane:



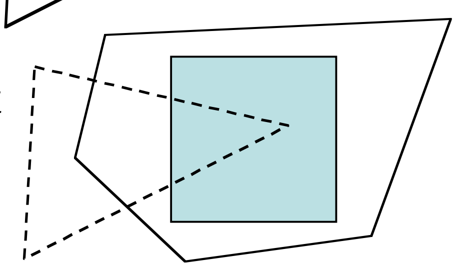
  - no surfaces project to the region



  - only one surface completely surrounds the region



  - only one surface is completely inside the region or overlaps the region



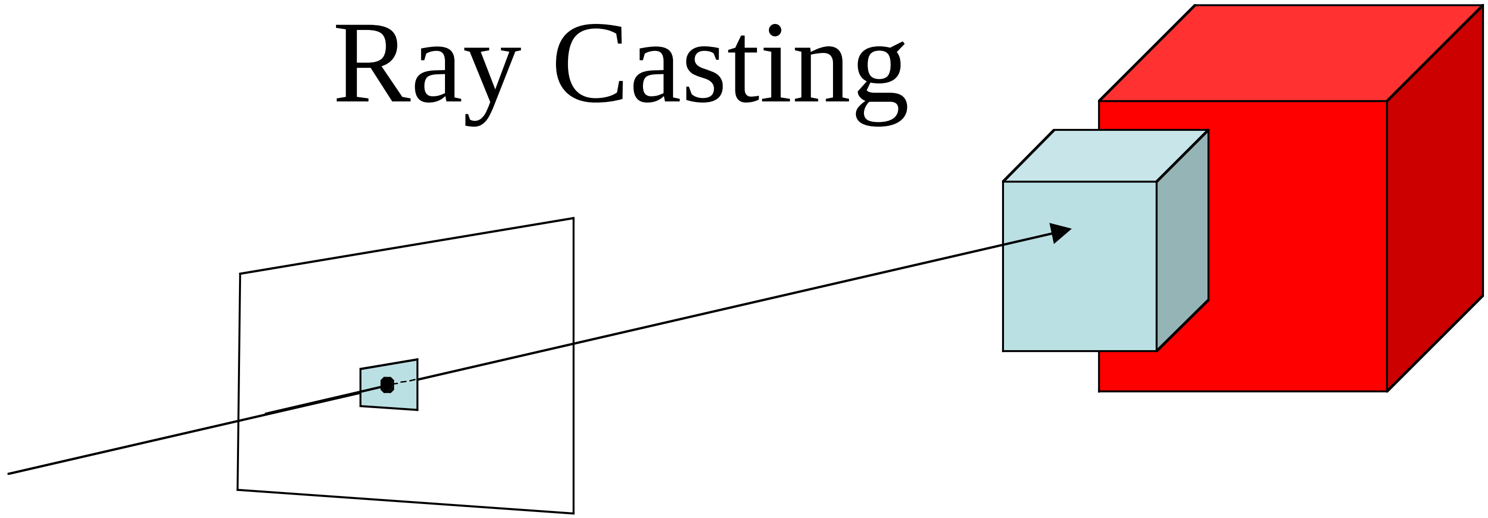
  - a polygon is completely in front of everything else in that region (consider depth of the polygons at the corners of the region)

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.

- Advantages:
  - can be very efficient
  - no over rendering
  - anti-aliases well

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.
- See §13.4 for discussing regarding intersection with spheres in perspective space.
- For polygons, we can use the standardized orthographic space where we can work in 2D.
- Useful for "picking"--not expensive here (why?)