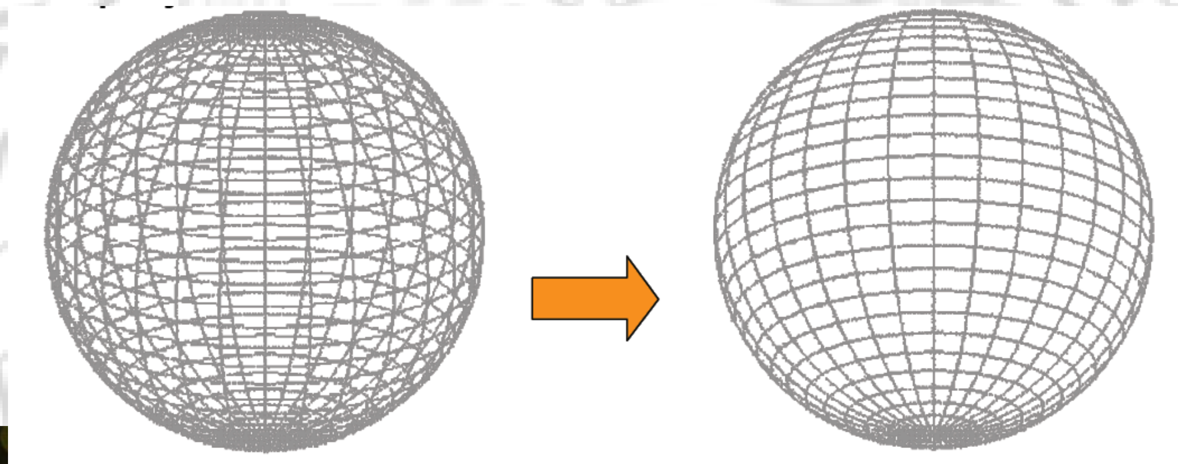
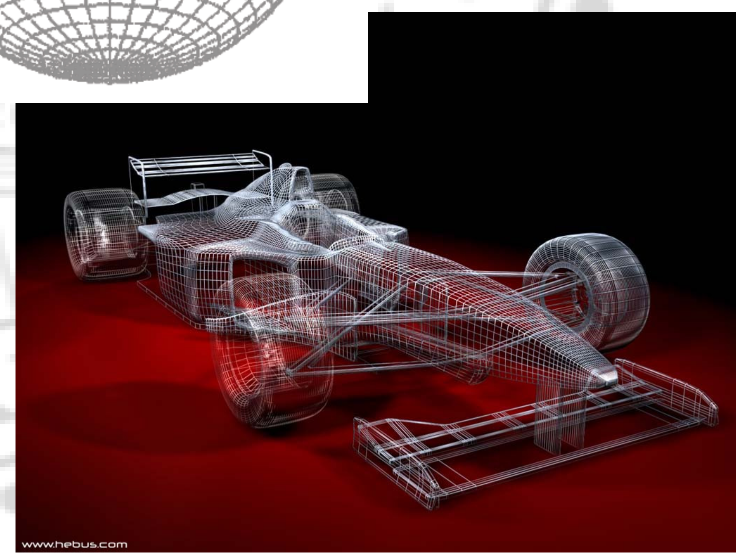
Visible Surfaces Determination





HLHSR

- Assumptions
  - objects are made of polygonal patches
  - all patches are opaque
- Goal
  - visibility from a view point

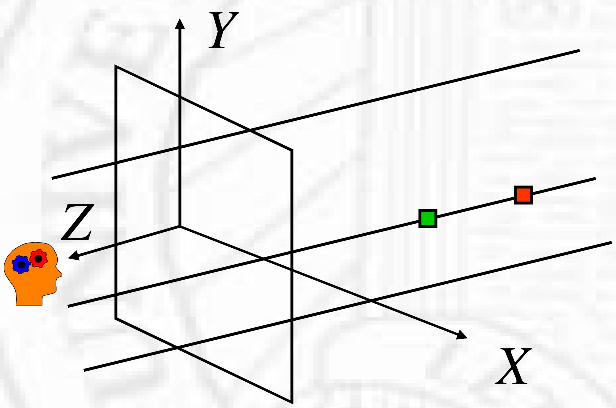
OpenGL

- Very simple, you need to do only two things
  - Prepare buffer
    - glutInitDisplay(GLUT_DEPTH | …);
      - *distance* to the view point is recorded
    - glClear(GL_DEPTH_BUFFER_BIT);
      - clear to the far clipping plan distance (1.0)
  - Enable depth comparison
    - glEnable(GL_DEPTH_TEST);
  - Tell OpenGL how to do the depth comparison
    - glDepthFunc(); default is GL_LESS (in *front* of the far clipping plane)
    - Visible z values are *negative*, but distance (depth) is *positive*

Depth comparison

- at 3D
- before projection
- after modeling, normalization, etc. transform

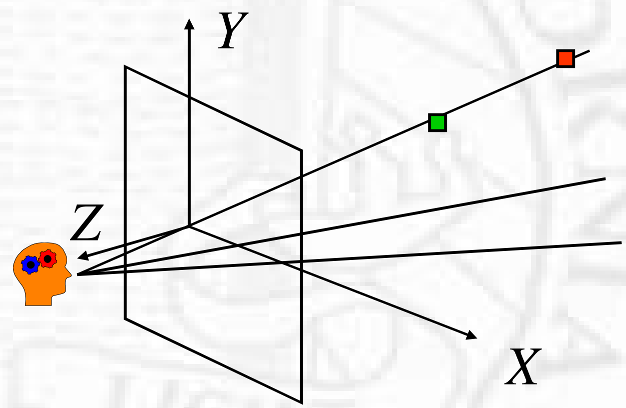
❖ Parallel



$$if\,(X_1 == X_2\ \&\&\ Y_1 == Y_2)$$

compare Z_1 *and* Z_2

❖ Perspective



$$if\,(\frac{X_1}{Z_1} == \frac{X_2}{Z_2}\ \&\&\ \frac{Y_1}{Z_1} == \frac{Y_2}{Z_2})$$

compare Z_1 *and* Z_2

General Approaches

- *Image* Space

  for (each pixel in the image) {

  determine the object closest to the viewer

  draw the object at that particular pixel }
- *Object* Space

  for (each object in the scene) {

  determine which parts of the object whose views are unobstructed

  draw those parts of the object }
- Hybrid

Useful techniques

- Coherence
  - parts of an object or an environment exhibit local similarity
- Bounding volumes
  - simplifies intersection tests
- Hierarchy
  - e.g., hierarchical bounding volumes
- Spatial partitioning
  - exploit spatial coherency
- Back face culling
  - e.g. convex objects viewing from outside

Scan-line Algorithm

- Image space
- Exploit scan-line coherency
- Image is generated one scan line at a time
  - keep all active polygons (edges)
  - determine their Z-ordering
  - ordering can change when cross edges

❖ Edge table

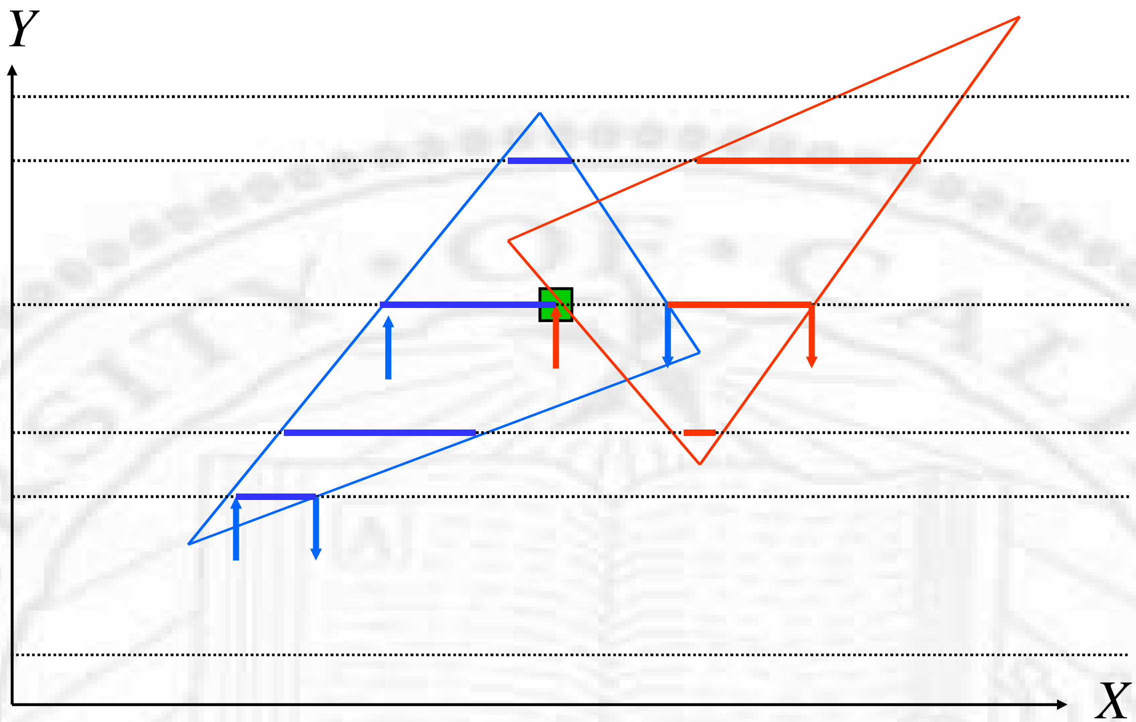
- sorted by smallbest y into buckets (scan lines)
- x coor at smallest y
- y coor at largest y
- x increment
- polygon ID

❖ Polygon table

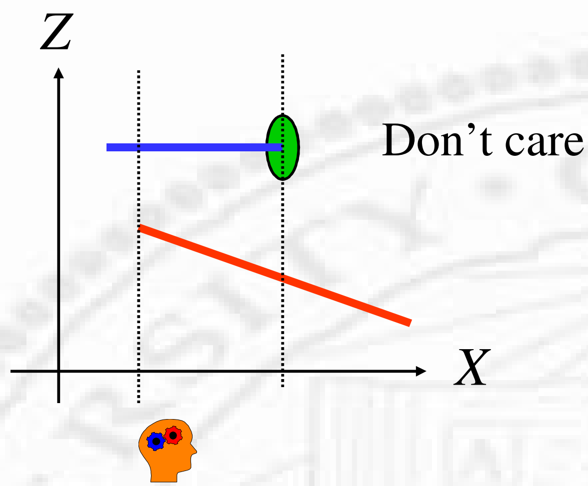
- coefficients of plane equation
- shading and color info
- IN/OUT flag

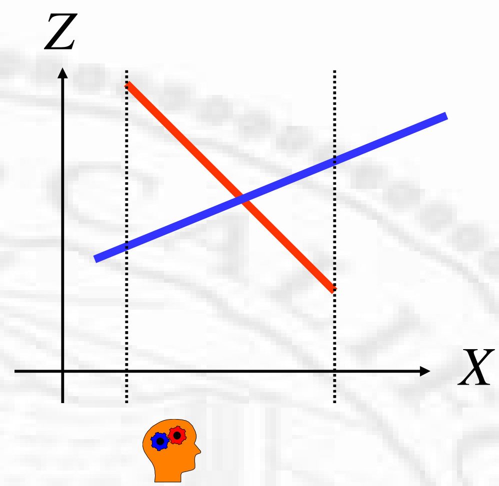
❖ Active Edge table

- At each scan line
- delete edges no long intersect current scan
- update x coordinates of active edges
- add new edges from the current bucket
- sort x coordinates of all active edges



- Scan in none - background
- Scan in one - paint
- Scan in multiple - order test





- Non-intersecting polygons
  - order change at edges
  - re-compute when scan leaving occluding polygons
- Intersecting polygons
  - order change at edges & *intersections*
  - splitting polygons may be necessary

Z Buffer

- Simplest (and most widely used) Object space
- Amenable to hardware implementation

```
Initialization
    ZB<- most distant Z;
    IB <- background color;
for each polygon {
    for each pixel in polygon {
        compute Z(x,y);
         if Z(x,y) is closer then ZB(x,y) {
              ZB(x,y) = Z(x,y);
              IB(x,y)  = polygon color;
        }
    }
}
```

Depth-Sort (List-Priority)

❖ Hybrid

Initialization

sort polygons in order of decreasing distance

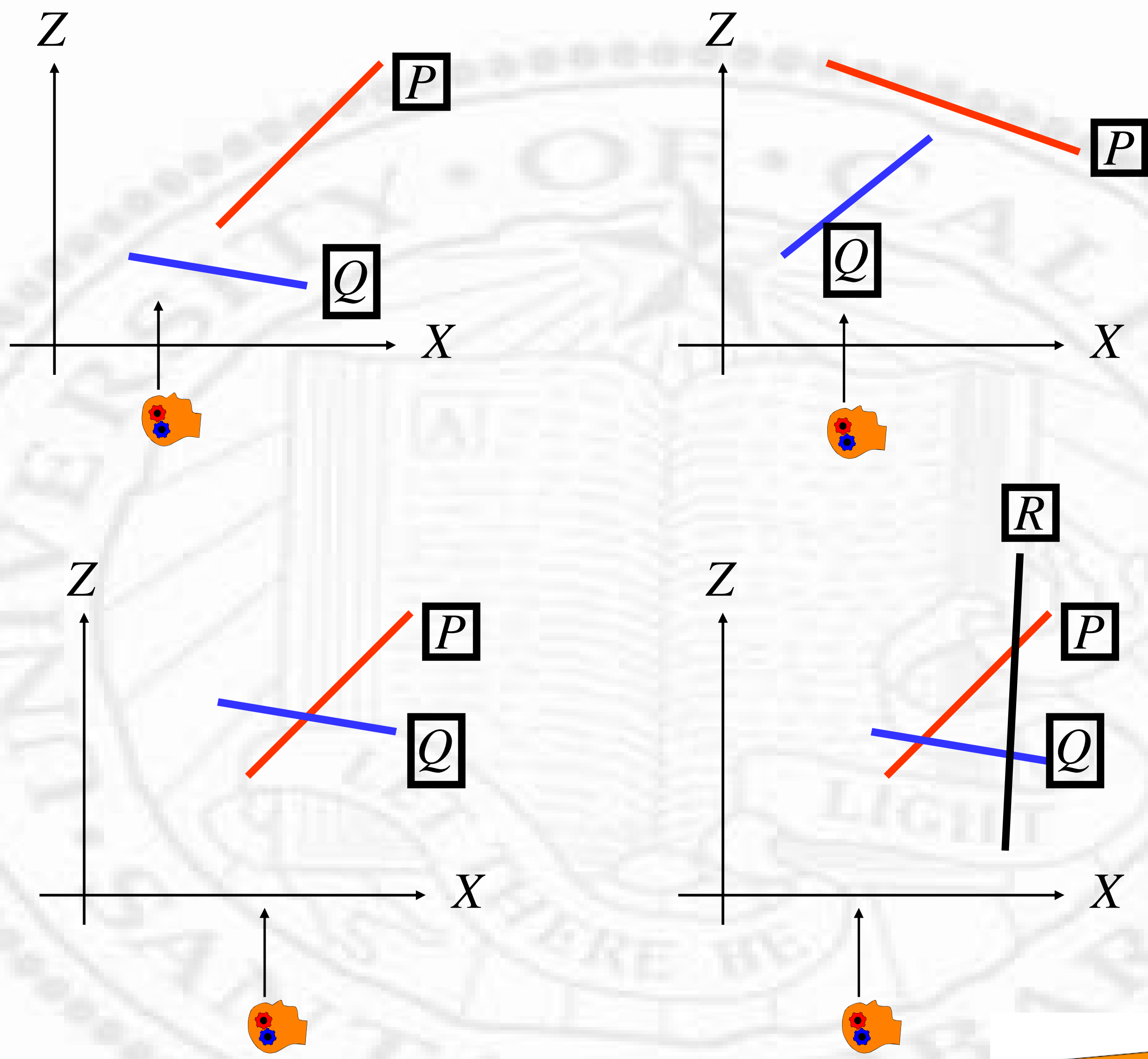
Resolve Ambiguity

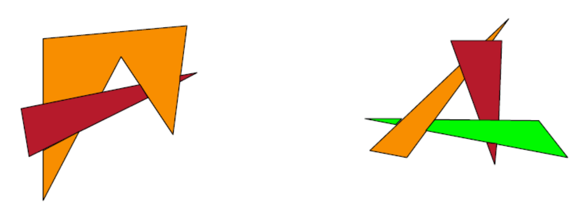
Pick the polygon *(P)* at the front (most distant) of the list
and for all polygon *(Q)* whose Z-extent overlap that of *P*'s

1. Do the polygons' x extent not overlap?
2. Do the polygons' y extent not overlap?
3. Is *P* entirely on the opposite side of *Q*'s plane from the viewpoint?
4. Is *Q* entirely on the same side of *P*'s plane as the viewpoint?
5. Do the projections of the polygons not overlap?

Switch *P* and *Q* if all above fail

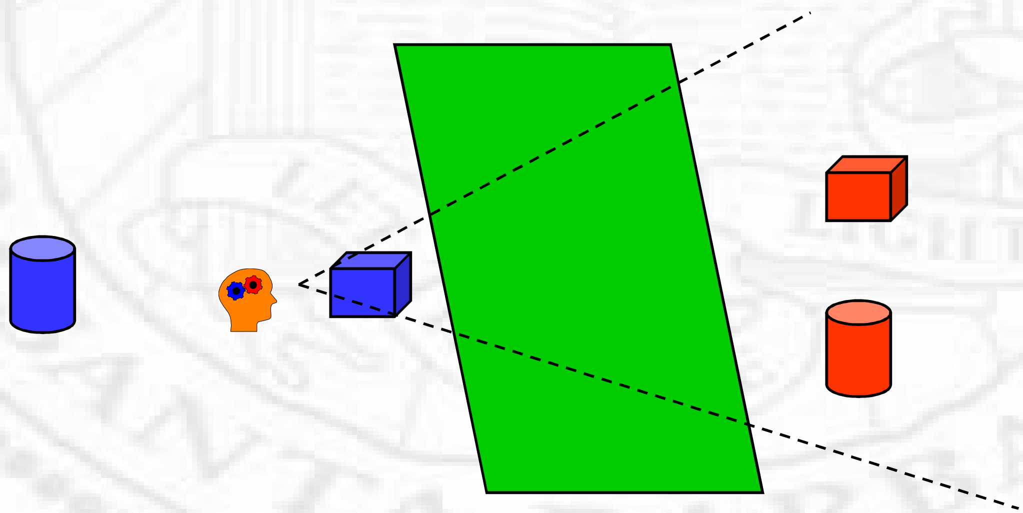
Scan Conversion


Z
P
Q
X
Z
P
Q
X
R
Z
P
Q
X
Z
P
Q
X



BSP-Tree (Object Space)

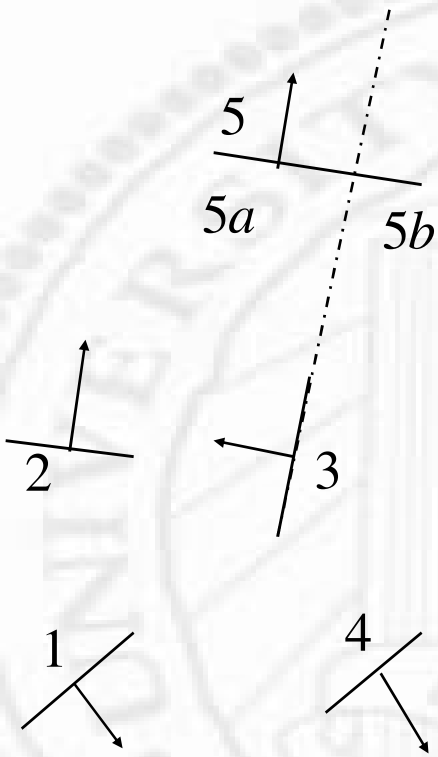
- Based on a simple observation
  - if the space is divided in half, then polygons on the side that does not contain the observer cannot obscure polygons on the same side as the observer

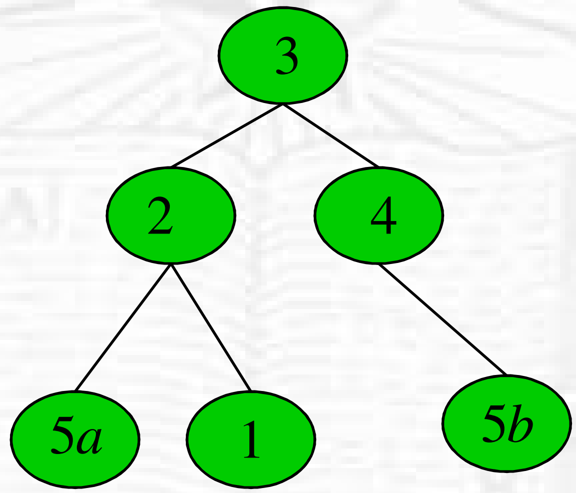


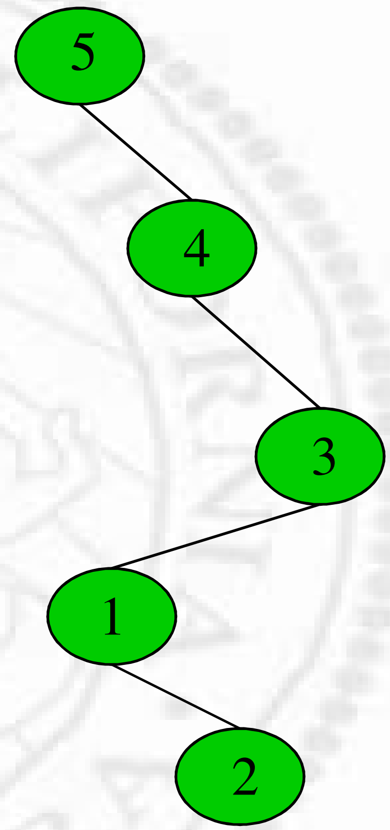
BSP Tree (cont.)

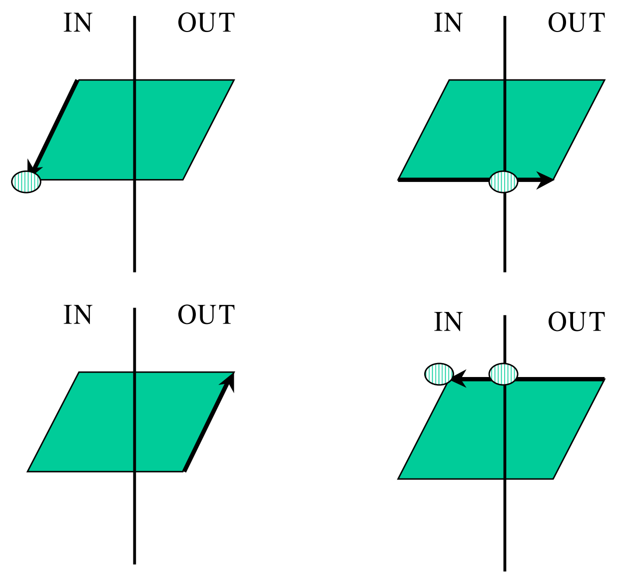
- Record the spatial adjacency info in a tree
  - choose a scene polygon and use its plane to partition the space into two halves
  - scene polygons are put in one half
  - *split polygons that straddle the partition plane*
  - recursively apply the algorithm until no more polygons can be used
  - good for
    - static scene structures
    - moving observer locations
    - e.g. fly-through, walk-through

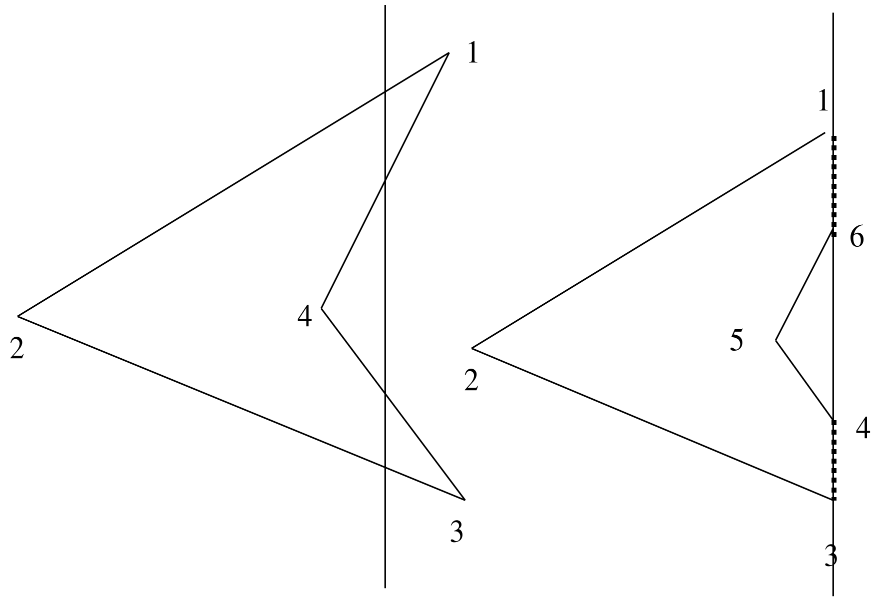
Example of BSP Tree











Original

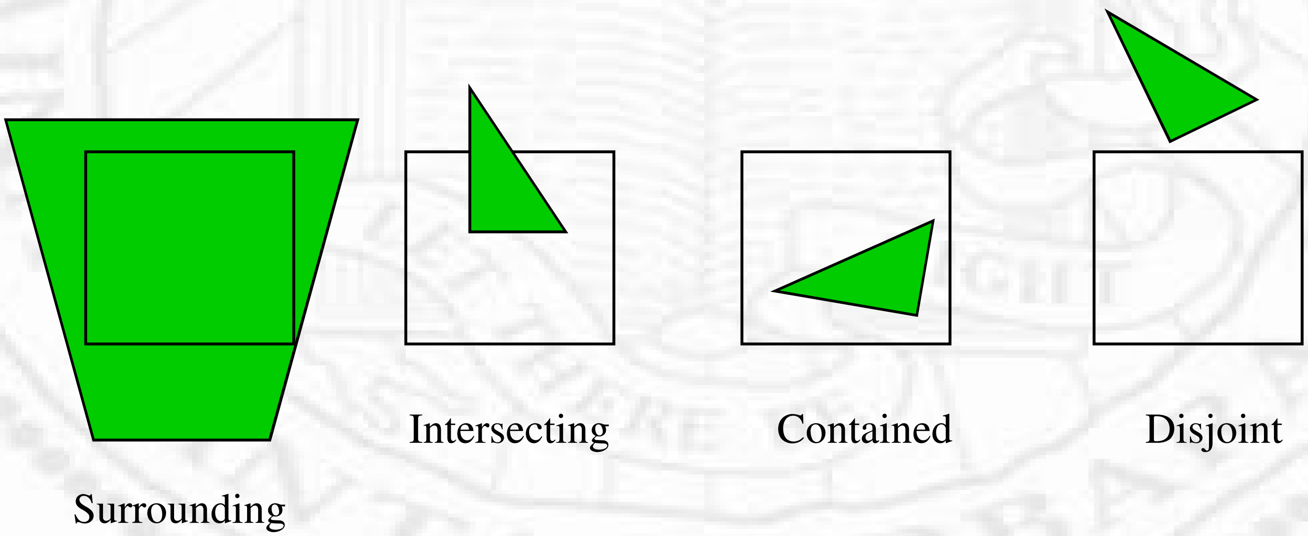
- Iterate through the list of vertices
- Output vertices based on IN/OUT relations

Rendering

- An in-order traversal
- At each node
  - traverse the subtree not containing the observer
  - render polygons at the node
  - traverse the subtree containing the observer

Area Sub-division

- Divide-and-Conquer
- Divide an image region until it is easy to decide which polygon or polygons are visible



Area Sub-division

- Disjoint
  - render background color
- One intersecting or contained
  - render background then polygon
- One surrounding
  - render polygon color
- More than one surrounding, intersecting, contained
  - if the surrounding polygon is in front
- Otherwise, recursive subdivision

Visible Surface Ray Tracing

```
for (each scan line) {
    for (each pixel in scan line) {
        compute ray direction from COP (eye) to pixel
        for (each object in scene) {
            if (intersection and closest so far) {
                record object and intersection point
            }
            set pixels color to that at closet object intersection
        }
    }
}
```

Compute Intersection

- A (low-order) implicit representation (f(x,y,z) =0) can be useful
- >0 (outside), =0 (surface), <0 (inside)
- Two examples
  - Spheres (implicit representation)
  - Polygons (parametric representation)

$$sphere: (X-a)^2 + (Y-b)^2 + (Z-c)^2 - r^2 = 0$$

$$ray: \begin{cases} X = X_o + t\Delta X \\ Y = Y_o + t\Delta Y \\ Z = Z_o + t\Delta Z \end{cases}$$

$$X^2 - 2aX + a^2 + Y^2 - 2bY + b^2 + Z^2 - 2cZ + c^2 - r^2 = 0$$

$$(X_o + t\Delta X)^2 - 2a(X_o + t\Delta X) + a^2 +$$
$$(Y_o + t\Delta Y)^2 - 2b(Y_o + t\Delta Y) + b^2 +$$
$$(Z_o + t\Delta Z)^2 - 2c(Z_o + t\Delta Z) + c^2 - r^2 = 0$$

$$(\Delta X^2 + \Delta Y^2 + \Delta Z^2)t^2 +$$
$$2\{\Delta X(X_o - a) + \Delta Y(Y_o - b) + \Delta Z(Z_o - c)\}t +$$
$$(X_o - a)^2 + (Y_o - b)^2 + (Z_o - c)^2 - r^2 = 0$$

$$(\Delta X^2 + \Delta Y^2 + \Delta Z^2)t^2 +$$
$$2\{\Delta X(X_o - a) + \Delta Y(Y_o - b) + \Delta Z(Z_o - c)\}t +$$
$$(X_o - a)^2 + (Y_o - b)^2 + (Z_o - c)^2 - r^2 = 0$$

$$At^2 + Bt + C = 0 \quad \Delta = B^2 - 4AC$$

$$t\begin{cases} \Delta > 0 & \text{insersecting} \\ \Delta = 0 & \text{grazing} \\ \Delta < 0 & \text{non insersecting} \end{cases}$$

$$plane : aX + bY + cZ + d = 0$$

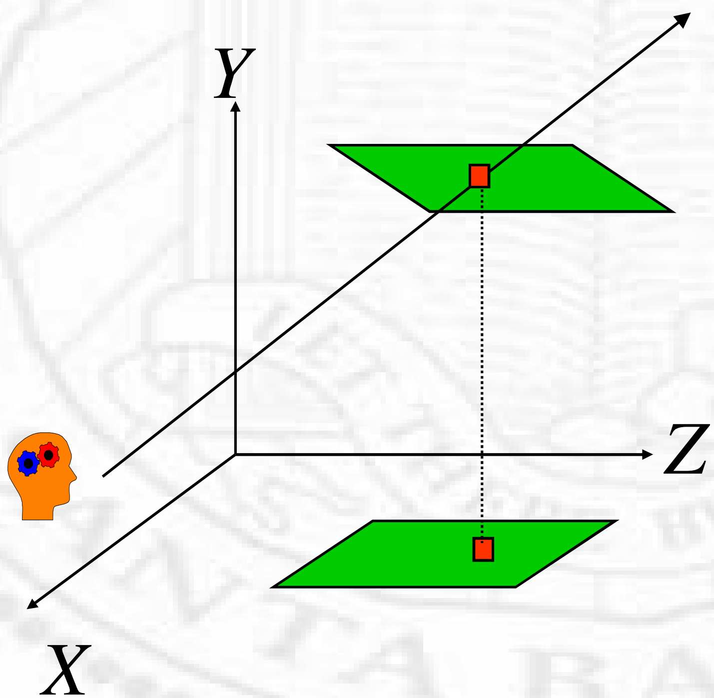
$$ray : \begin{cases} X = X_o + t\Delta X \\ Y = Y_o + t\Delta Y \\ Z = Z_o + t\Delta Z \end{cases}$$

$$a(X_o + t\Delta X) + b(Y_o + t\Delta Y) + c(Z_o + t\Delta Z) + d = 0$$

$$t = -\frac{aX_o + bY_o + cZ_o + d}{a\Delta X + b\Delta Y + c\Delta Z}$$

- There will be a reasonable t value, unless the denominator is zero (the line and the plane are parallel)
- But is the intersection point actually inside the polygon?

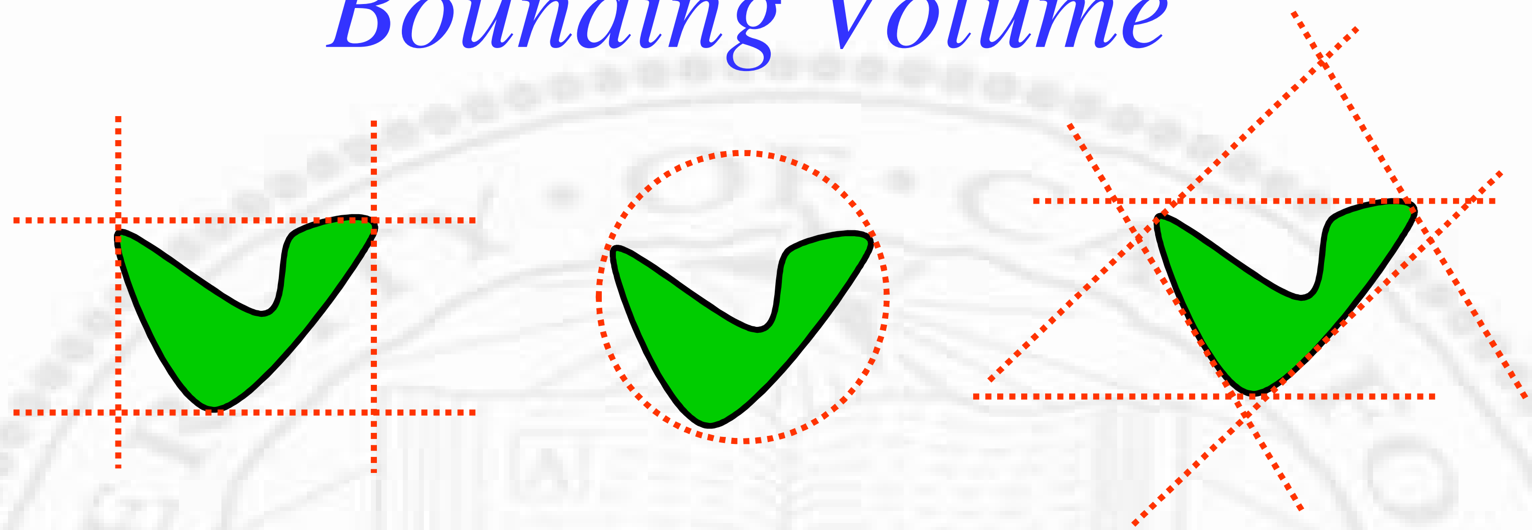
- Point-in-polygon test (point must be on the inside relative to all polygon edges)
- Can be done in 2D



Avoid intersection computation

- Bounding volume
- Object hierarchy
- Spatial partitioning

Bounding Volume



$$Slab: aX + bY + cZ + d = 0$$

$$ray: \begin{cases} X = X_o + t\Delta X \\ Y = Y_o + t\Delta Y \\ Z = Z_o + t\Delta Z \end{cases}$$

$$t = -\frac{aX_o + bY_o + cZ_o + d}{a\Delta X + b\Delta Y + c\Delta Z} = \frac{A + D}{B}$$

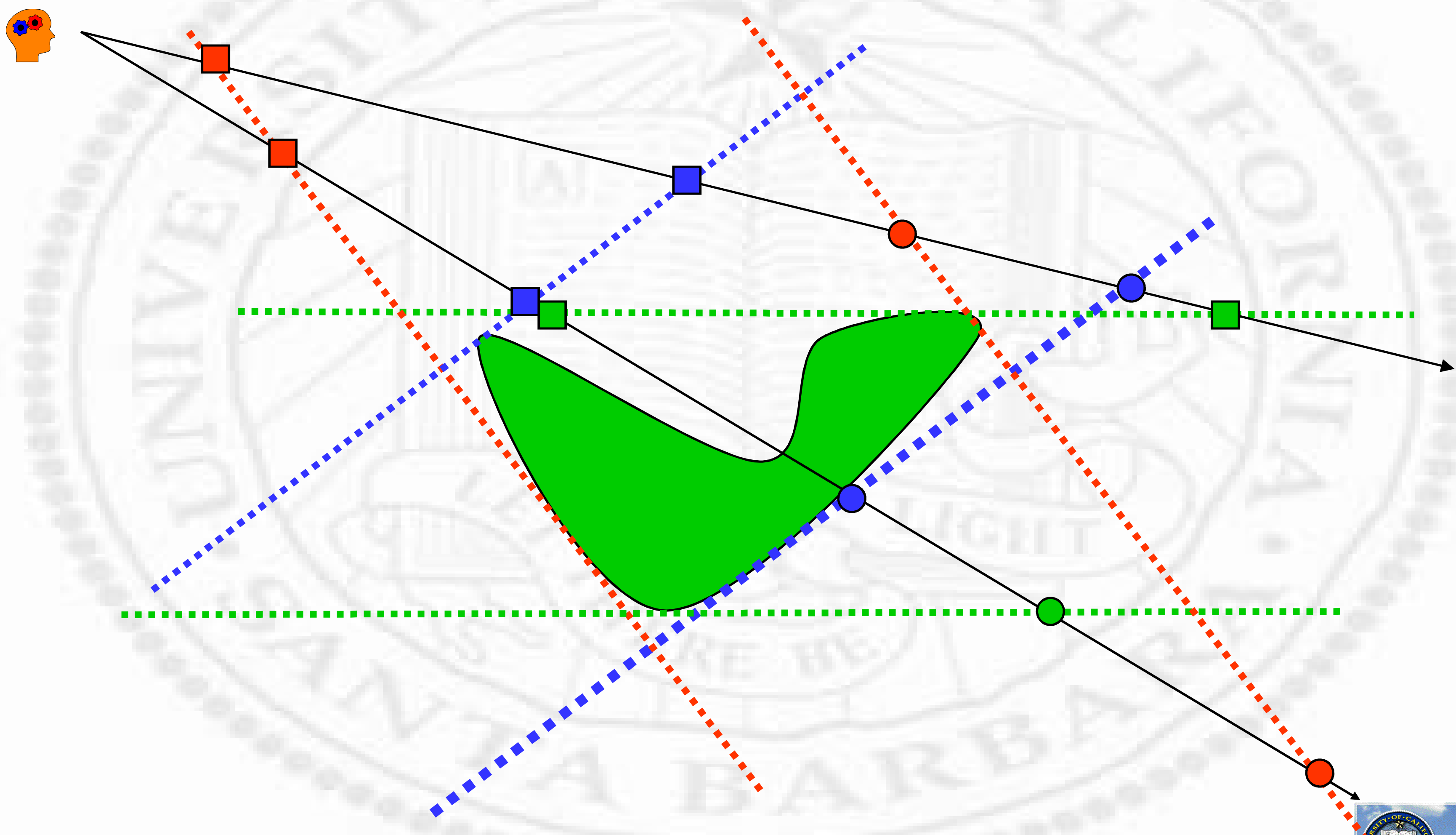
A: per ray per slab set

B: per ray per slab set

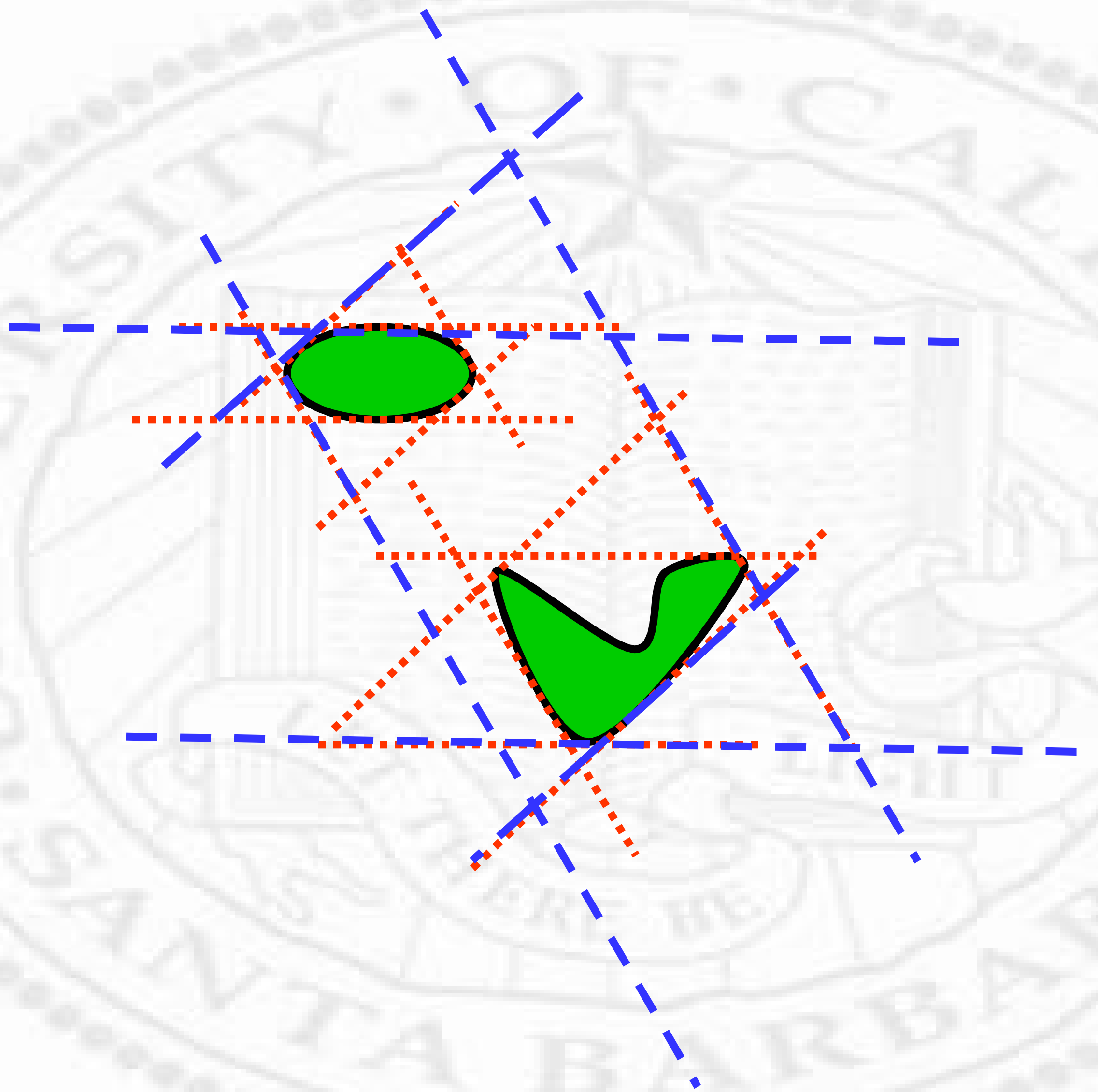
D: per slab

Bounding Volume (cont.)

- All the maximum (circle) intersections must be after all the minimum (square) intersections



Hierarchical Bounding Volume



Spatial Partitioning

- Ray can be advanced from cell to cell
- Only those objects in the cells lying on the path of the ray need be considered
- First intersection terminates the search