

CS3500 Computer Graphics

Visible Surface Determination

P. J. Narayanan

Spring 2009

Visible Surface Determination

- Multiple object points can project to the same image pixel in 3D. Which one gives the colour to the pixel?
- **Visible line/surface determination** or **Hidden line/surface elimination.**
- Consider objects to be drawn or pixels to be painted.
- Two parts: Visibility to view volume and mutual occlusion of objects.
- Need to analyze the scene from the camera.

Simplifying VSD

- Extents: If two object do not overlap in x and y extents, they cannot occlude each other. Bounding volumes must overlap for possible obstruction.
- Spatial Partitions: If the whole object overlaps, divide into sub-parts and check for each.
- Hierarchy: If higher levels of the hierarchy contains the lower ones entirely, checking can be done from top down.
- **Coherence** or local similarity in: objects, faces, edges, scan-line, area, depth, etc.
Results in incremental computation algorithms.

View Frustum Culling

- Eliminate objects that are outside the view volume.
- Large parts of the scene will be eliminated this way.

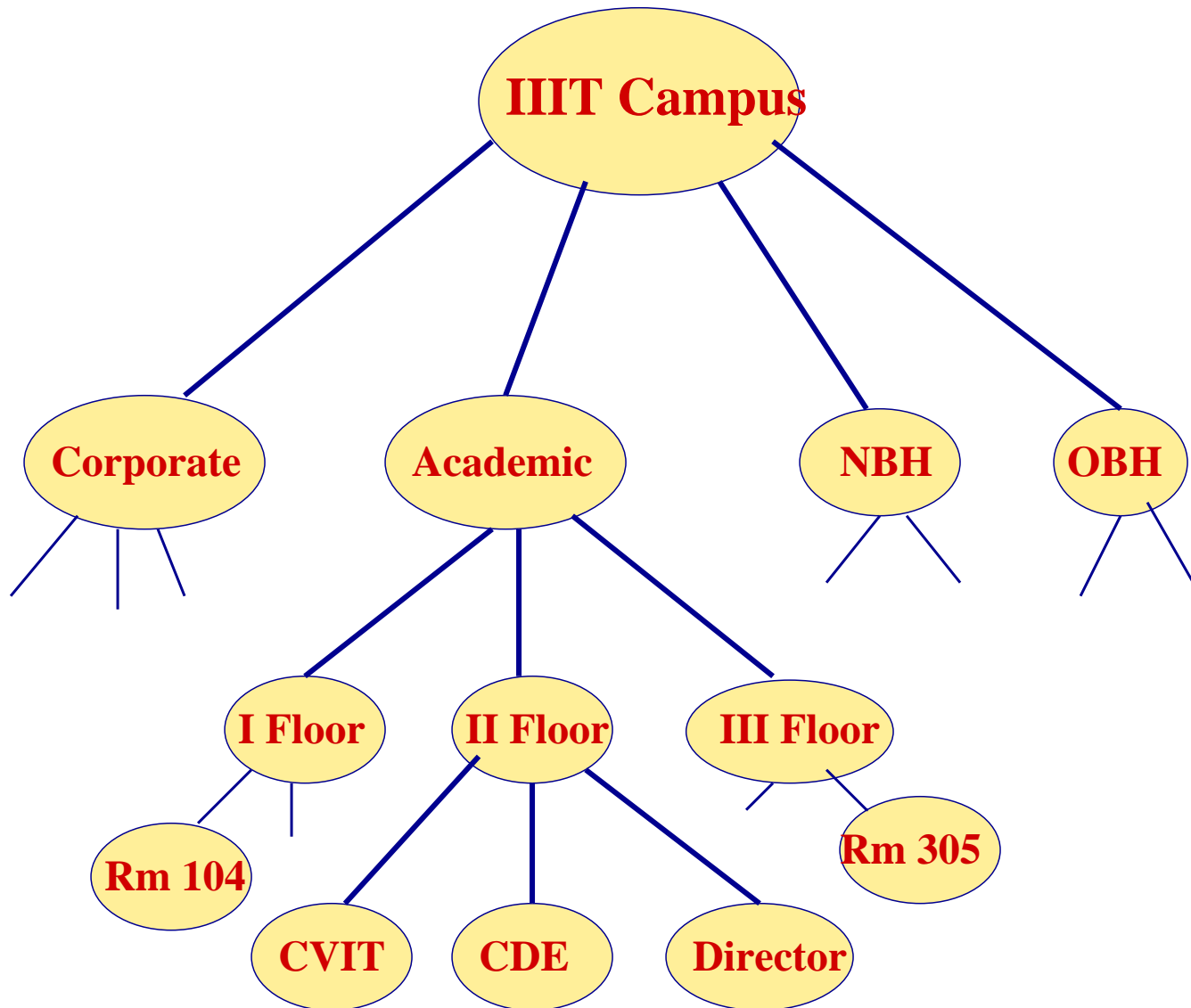
Compare bounding volume of `node` with view volume

If intersection is null, eliminate tree from root

If volume contained in frustum, render without clipping

Else, recursively check for each child of `node` till leaf

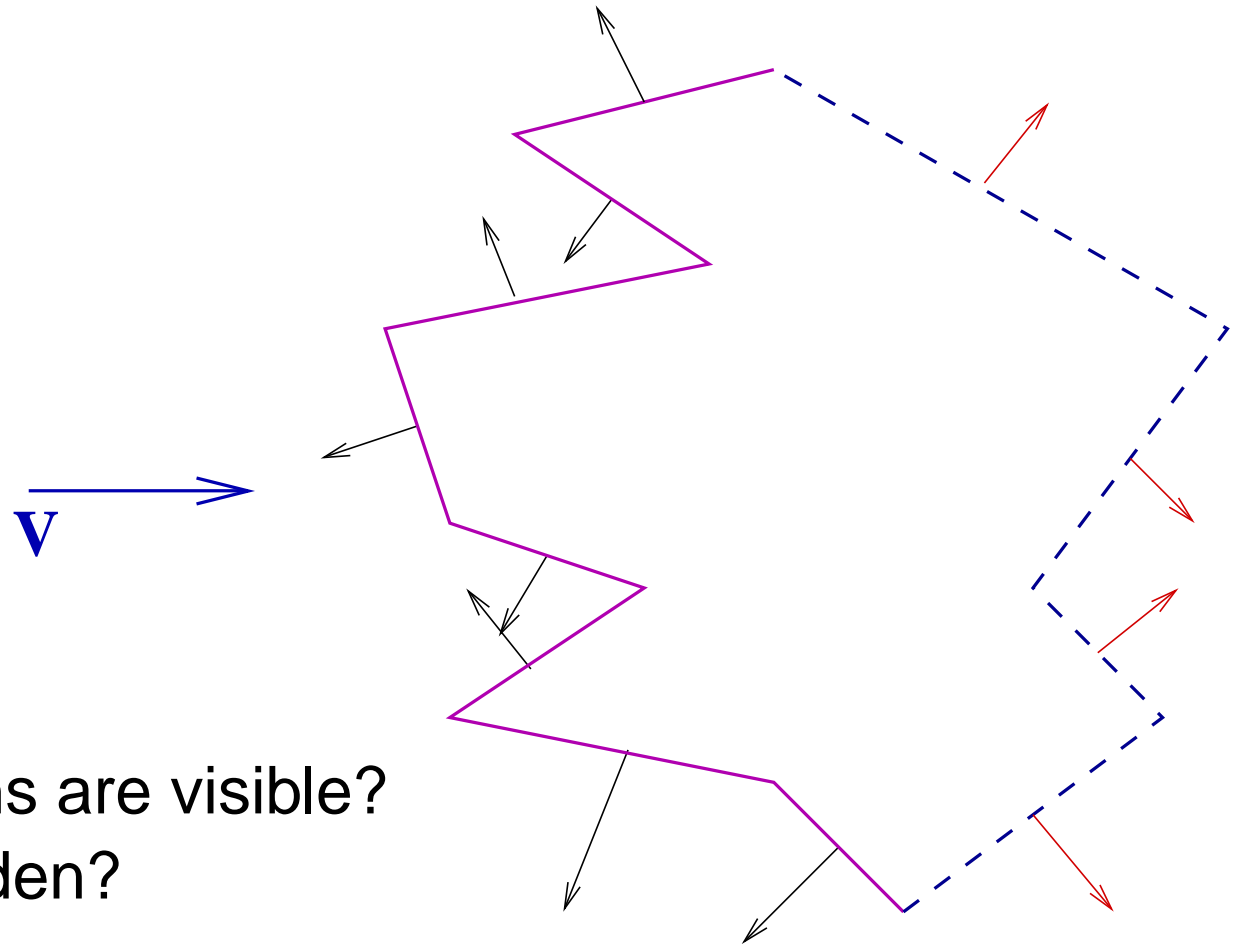
- Object hierarchy helps. Whole campus, each building, each floor, each room, etc., in the hierarchy.



Box-Frustum Intersection

- Orthographic: Intersection of two boxes. Perform by checking simultaneous X-Y-Z overlaps.
- Perspective: Clip against 6 planes of the frustum.
- Or, project. BBox becomes a frustum. Find the bounding box (conservative) of the new one.
- AABB: Axis-Aligned Bounding Box
OBB: Oriented Bounding Box.

Viewing a Solid Object



- Which polygons are visible?
Which are hidden?

Back-Face Culling

- Only the front side can be seen of solid, opaque objects.
- If surface normals point out of the object, polygons with normals pointing away from the viewer cannot be seen.
- If $\mathbf{v} \cdot \mathbf{n} < 0$, draw. Else discard. \mathbf{v} is a vector from CoP to any point on the polygon and \mathbf{n} the surface normal.
- After normalizing, DoP is $[0 \ 0 \ -1]^T$. Eliminate polygons with negative z component in the normal vector.
- Half the polygons eliminated on the average.
- If there is only one object, no other VSD necessary.

Object-Precision Algorithm

```
for each object in the world {  
    Determine unobstructed parts of the object  
    Draw those parts with appropriate colours  
}
```

- Complexity depends on the number of objects.
- If the image size changes, only the drawing step needs to be redone.
- Works in the original continuous object space.

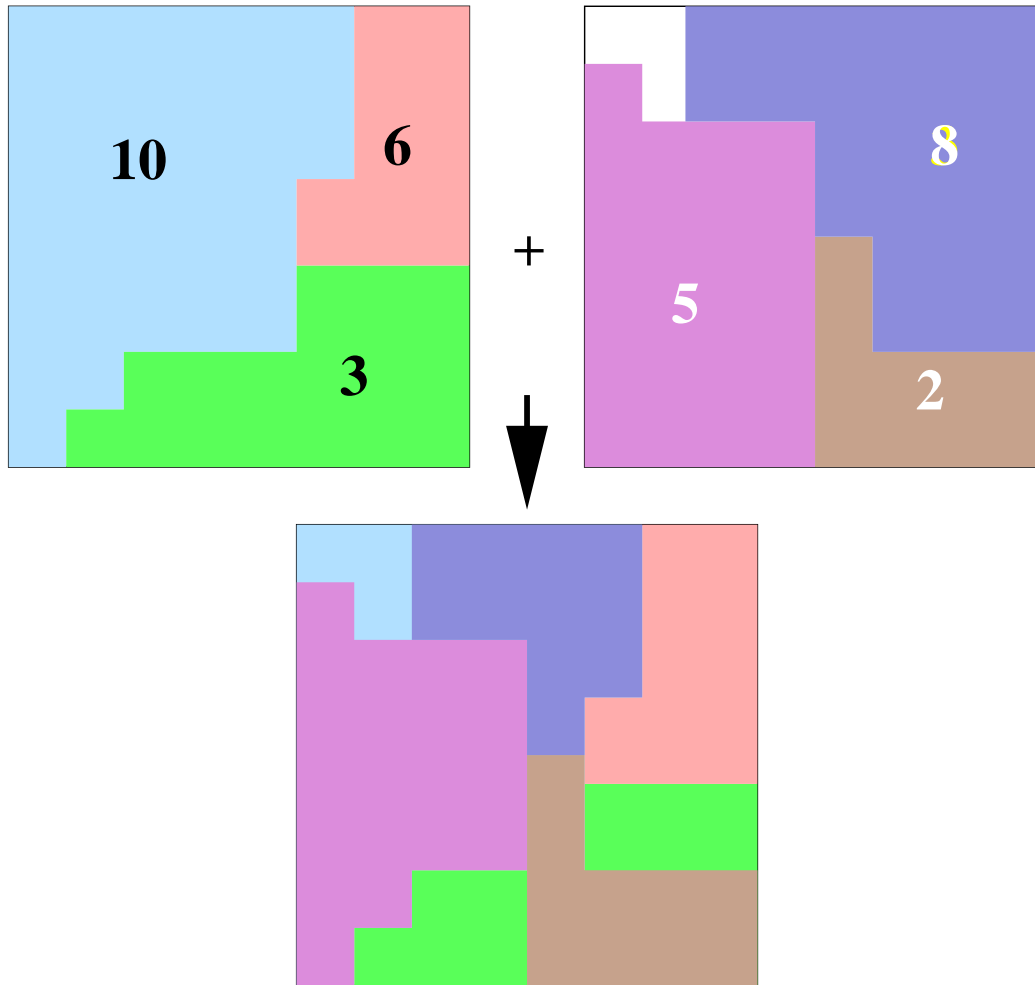
Image-Precision Algorithm

```
for each pixel in the image {  
    Determine closest object in the direction of projector  
    Draw the pixel with appropriate colours  
}
```

- Works in the discrete image space.
- Complexity depends on the number of pixels.
- A natural choice for raster graphics.
- Image resizing involves repeating the entire work.

Z-buffer or Depth-buffer Algorithm

- An image-precision algorithm that needs a z-buffer parallel to the frame buffer.
- z values after the normalizing transformation is stored into the z-buffer along with colour info to the frame buffer. We have $0 \geq z \geq -1$.
- Since depth ordering is maintained, a larger z implies a closer 3D point in any direction.
- Write to frame buffer and z-buffer only when the z value is larger than previously stored value.

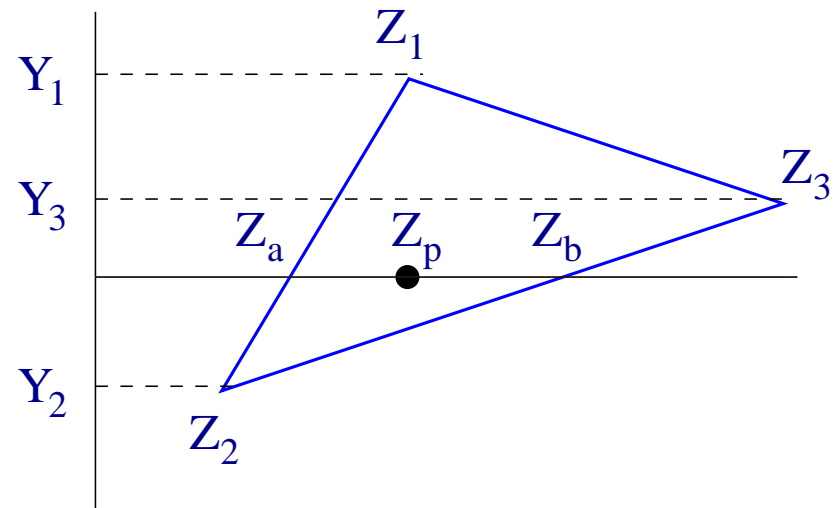


Pseudocode

```
for  $0 \leq y \leq Y_{\max}$   
    for  $0 \leq x \leq X_{\max}$   
        WritePixel (x, y, bgnd_colour)  
        WriteZ (x, y, -1)    // Farthest value  
  
for each polygon  
    for each pixel in polygon's projection  
        pz = polygon's  $z$ -value at pixel  $(x, y)$   
        if (pz > ReadZ(x, y))  
            WriteZ()  
            WritePixel()
```

Computing z for Interior Points

- Exploit coherence in depth values over the plane to compute interior z values given the vertex values.
- $z_a = z_1 + (z_2 - z_1) \frac{(y_a - y_1)}{(y_2 - y_1)}$
 $z_b = ??$ $z_p = ???$
- When x or y increments by 1 along a side or a scan line, z changes by a constant Δz .



Z-buffering: Discussion

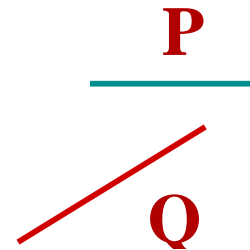
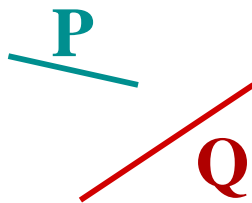
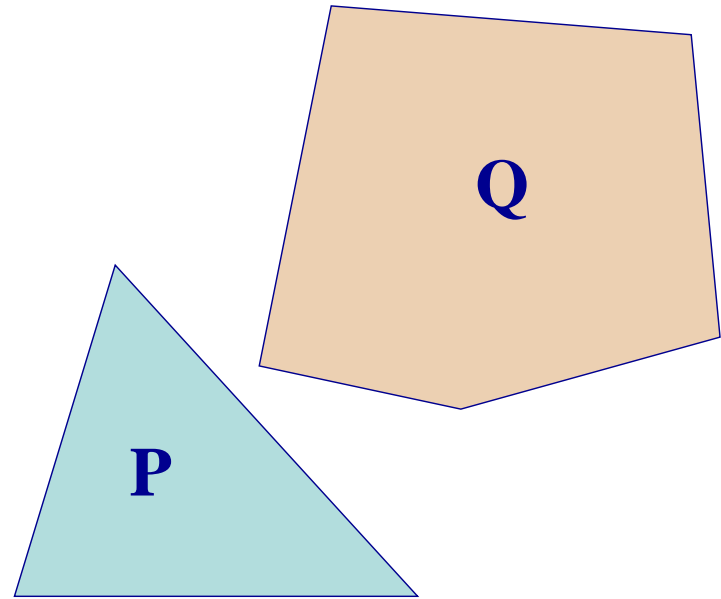
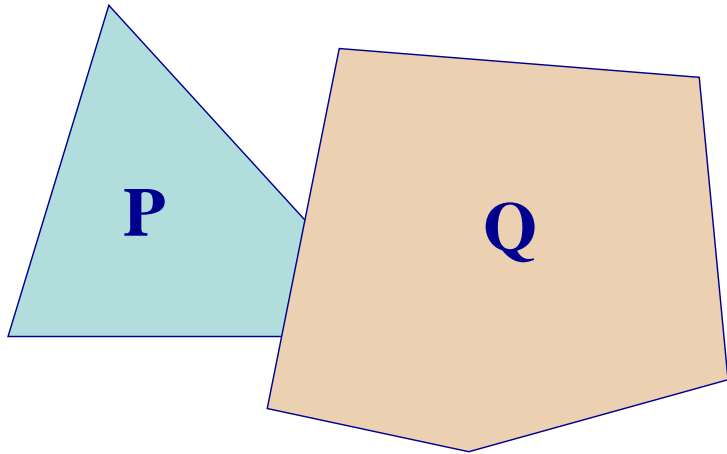
- Any shape with per pixel z can be handled correctly.
- Time is independent of number of primitives.
- Easy to implement; can do with a single scan-line Z-buffer.
- Z-buffer can be read back and saved.
- Needs extra memory, but memory is cheap.
- Can cause aliasing or **z-fighting** (*shimmering*).

List Priority Algorithms

- Reorder objects such that the correct picture results if you draw them in that order.
- Example: If objects do not overlap in z , draw them from back to front.
- Objects may need splitting if no unique ordering exists.
- Ordering and splitting polygons: Object-precision operation.
- Overwriting farther points while scan conversion: Image-precision operation.

Depth-Sort or Painter's Algorithm

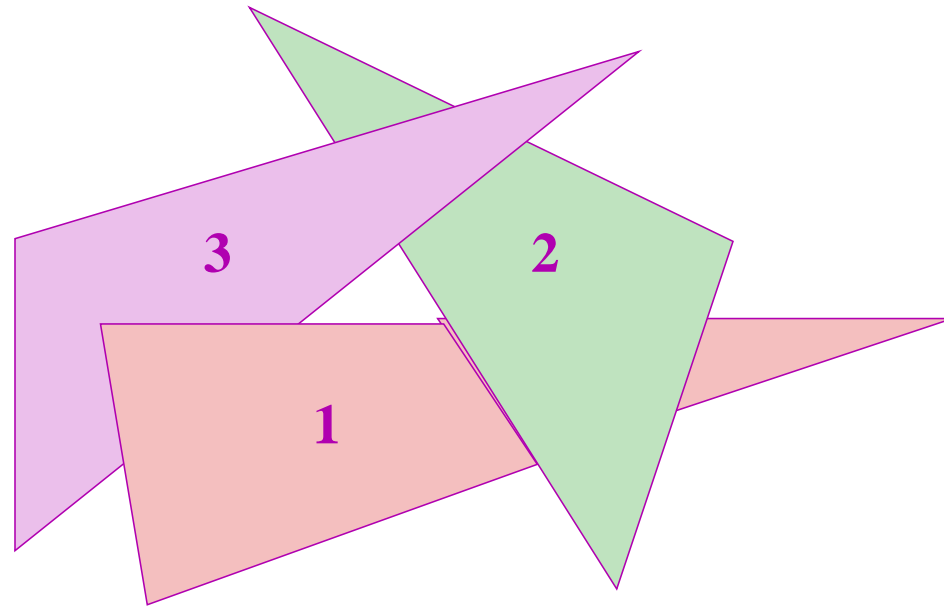
1. Sort polygons on the farthest z coordinate of each.
 2. Resolve ambiguities due to overlap in z .
Split polygons if necessary.
 3. Scan convert from **back to front**.
- Draw in the order of minimum z if no depth ambiguity.
Painter's Algorithm: Paint close objects over farther ones.
 - Each object can be thought of as belonging to a plane of constant z .



Ordering a Pair of Polygons

- Let P be the farthest polygon in the sorted list. If it doesn't overlap in z with any other polygon, it can be drawn immediately. Else let Q overlap with it in z .
- Test the following:
 1. Are P and Q non-overlapping in x ?
 2. Are P and Q non-overlapping in y ?
 3. Is P entirely on the opposite side of Q 's plane from viewpoint?
 4. Is Q entirely on the same side of P as the viewpoint?
 5. Are XY projections of P and Q non-overlapping?

- If the answer to any question is a **yes**, P can be drawn (i.e., scan converted) before Q.
- Otherwise, reverse the roles of P and Q and check modified conditions 3' and 4'. If any is satisfied, move Q to the end of the list and make it P and go to the beginning. (Need to check Q against polygons P was farther to earlier!)
- Check conditions 3' and 4' only of P has not been moved to the end of the list earlier. (To avoid looping.)
- Otherwise, split P with the plane of Q (or vice versa). Delete P from the list. Insert the 2 new polygons in correct z order and proceed.



- No ordering can solve the ambiguities.
- Split polygon 1 to 1a (left) and 1b.
- Draw in the order: 1b, 2, 3, 1a.

Depth-Sort Algorithms: Discussion

- Ensures back-to-front ordering for proper rendering.
- No aliasing effects introduced as objects are reordered/split.
- Reordering and splitting of polygons have to be done at run time.
- Redo the whole calculations if the view-point changes.
- Computationally expensive.

Binary Space Partitioning Trees

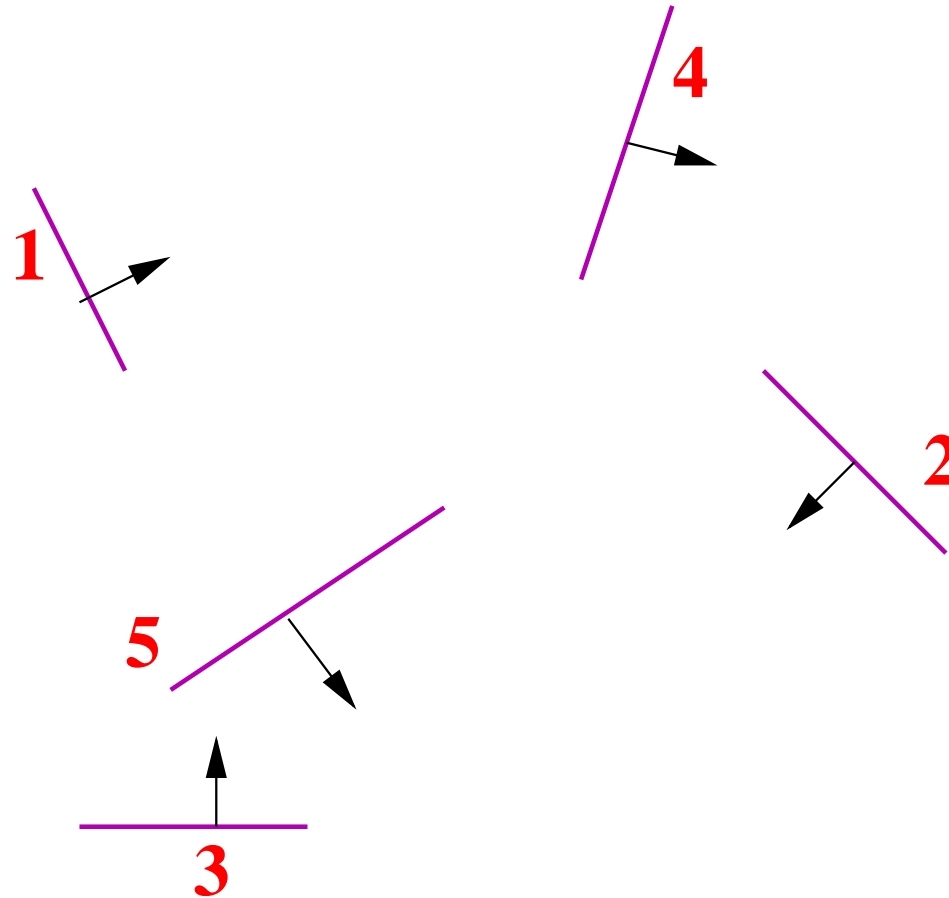
- Can we have simple linear display algorithm, perhaps using expensive preprocessing?
- Consider a plane in space that divides scene into two halves.
- Objects on the same side of the plane as the eye cannot be blocked by objects on the other side.
- Each side of the plane can further be divided using other planes till we reach a single object.

- If environments consist of clusters of objects, separate them using an appropriate plane.
- We end up with the **BSP Tree** representation of the scene.
- Internal nodes contain partitioning planes; leaf nodes are polygons.
- Some preprocessing to construct the tree, but simple algorithm to render using it from any viewpoint.

BSP Trees

- Use planes of polygons in the scene as partitioning planes.
- Normal direction indicates the “front” side of the plane.
- Each plane divides space into two sides.
- If a polygon lies on both sides of the plane, divide it into two parts.
- Continue this recursively till each side contains exactly one polygon.

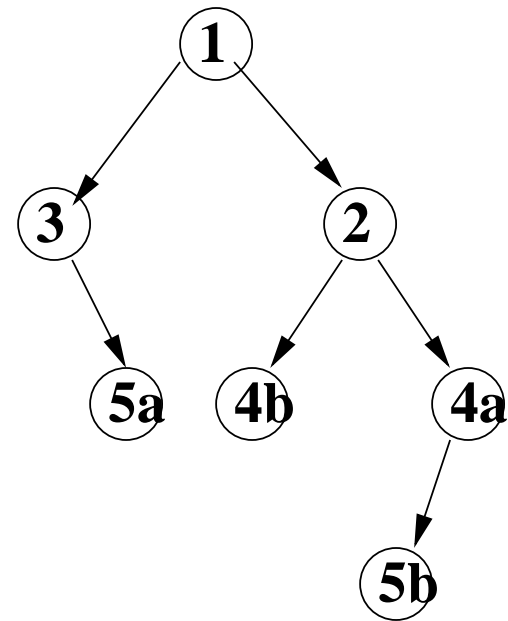
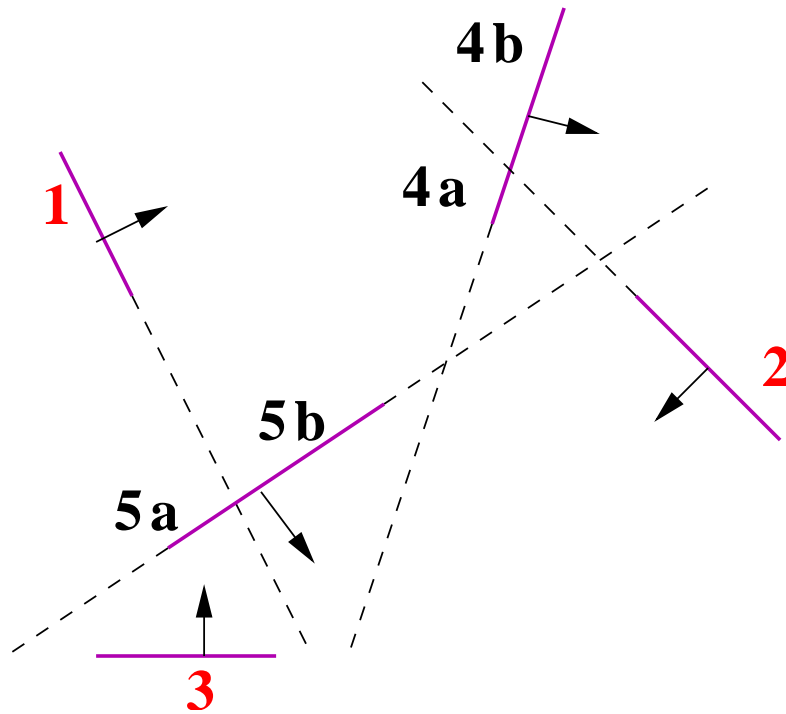
Example: BSP Tree



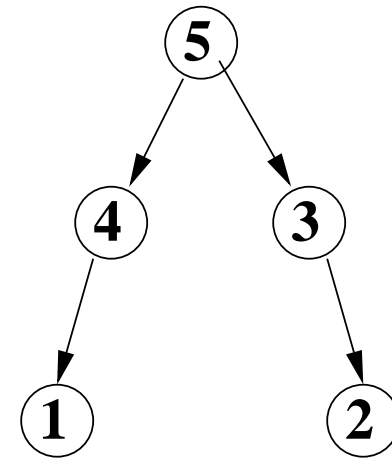
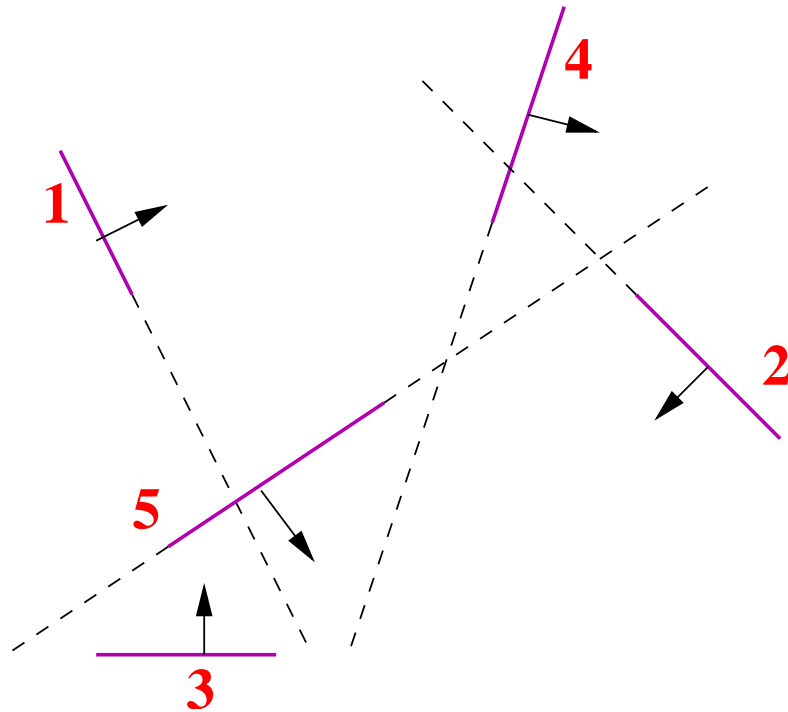
Pseudocode: BSP Tree Construction

```
makeBSPTree(pList)
  if (pList is empty)      return NULL
  root ← selAndRemove(pList); bList, fList ← NULL
  for each polygon p in pList
    if (p is in front of root)      addToList(p, fList)
    elsif (p is in back of root)    addToList(p, bList)
    else
      splitPoly(p, fp, bp)
      addToList(fp, fList); addToList(bp, bList)
  return combineTree(makeBSPTree(fList),
                    root, makeBSPTree(bList))
```

Example: Smallest First



Example: Largest First



Pseudocode: Displaying a BSP Tree

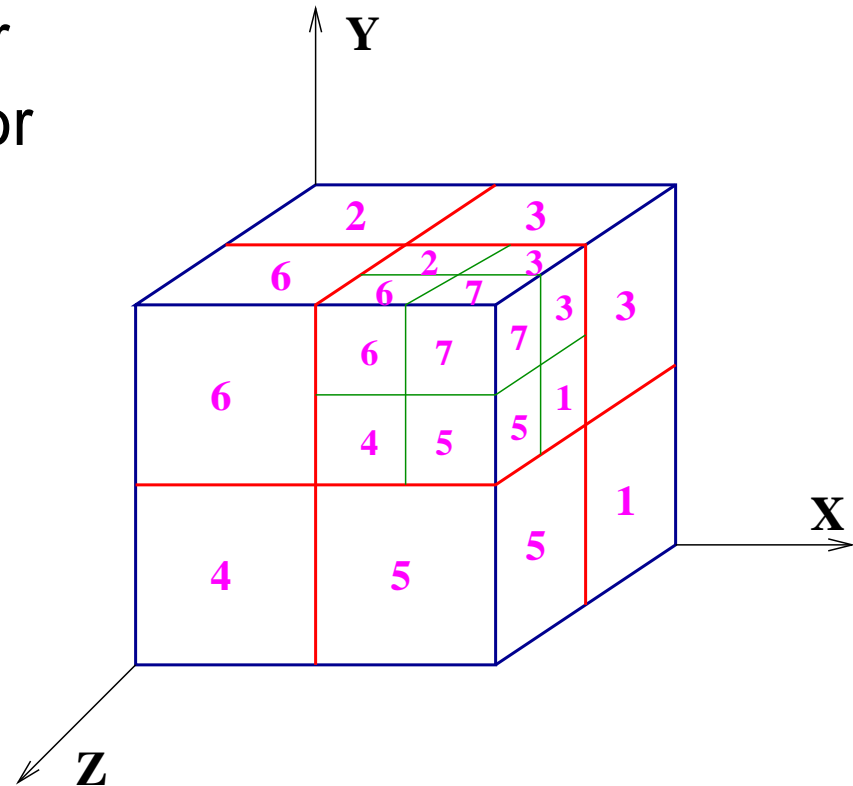
```
displayBSPTree(bTree)
  if (empty(bTree)) return
  if (eye in front of root)
    displayBSPTree(bTree→bChild)
    displayPoly(root)
    displayBSPTree(bTree→fChild)
  else
    display(BSPTree(bTree→fChild)
    if (no back-face culling)    displayPoly(root)
    display(BSPTree(bTree→bChild)
```

Performance Considerations

- Construction of the BSP tree is expensive.
- No splitting while rendering; everything is done during preprocessing.
- Straightforward display algorithm. Back-face culling woven into it.
- Strategy of selecting the root has a great impact.
- Select the polygon that splits least number of polygons.

Octree Visibility Algorithms

- Strict ordering of octants for visibility exists for octrees for orthographic projection, given the VPN (View Plane Normal).



- Only 3 of the faces can be visible from any viewpoint.

- If VPN is (+, +, +): **0, 1, 2, 4, 3, 5, 6, 7**
 - If VPN is (-, +, +): **1, 0, 3, 5, 2, 4, 7, 6**
 - If VPN is (+, +, -): **4, 0, 5, 6, 1, 2, 7, 3**
- Visit the octree in the back-to-front order, given the orientation of the camera with respect to the scene.
 - Holds good for perspective projections also if scene is only on one side of the plane.

Octree Visibility: Procedure

- **Preprocessing:** Divide the bounding volume into a octree hierarchy till each cell is (a) contains only 1 object and back-face culling will do **or** (b) contains a few objects so that depth-sort algorithm is fine **or** (c) we know how to handle all objects in some way.
- **Rendering:** Based on the view-plane normal or view direction. traverse the octree nodes recursively in a particular order. Back-to-front will result.

Ray Tracing for Visibility

- Perform exactly what our image-precision algorithm described.
 - for each pixel in the image**
 - Determine closest object in the direction of projector
 - Draw the pixel with appropriate colours
- Send rays from CoP through each image pixel to the world.
- Called **ray tracing** or **ray casting**.
- Equation of the ray is known. Need to intersect it with objects in the world.

Ray Equation

- If the CoP is (x_0, y_0, z_0) and pixel point is (x_1, y_1, z_1) , the ray is given by $(x_0 + t\Delta x, y_0 + t\Delta y, z_0 + t\Delta z)$, $t > 0$
- Negative values of t line behind CoP. $t = 1$ is at the projection or pixel plane.
- If the scene is in front of the plane, the region of interest is $t > 1$.
- Compute intersections with other objects. Closest object is the one with the smallest t value.

Intersection with Polygons

- Plane of the polygon is given by $Ax + By + Cz + D = 0$
- Intersection point: $t = -\frac{Ax_0 + By_0 + Cz_0 + D}{A\Delta x + B\Delta y + C\Delta z}$
- Does it lie within the polygon?
- Project to a coordinate plane and check for 2D polygon containment.
- Use the plane with largest area. This is determined by the largest absolute value of A, B, C .

Intersection with a Sphere

- Sphere is given by $(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2$.
- Substituting, we get: $(\Delta x^2 + \Delta y^2 + \Delta z^2) t^2 + 2[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] t + (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0$
- A quadratic equation. Solve for t . Real solution with smaller positive t is the one of interest.
- Can normalize such that the coefficient of t^2 is 1, since we are interested only in the relative values of t .

Ray Tracing: Discussion

- Ray tracing is an image precision operation.
- Intersection calculations need to be performed for each ray with each object. **Very expensive.**
- Hierarchical organization and spatial partitioning of data can help reduce the number of intersections computed.
- Coherence in image space enables us to trace a bunch of rays (**beams**) to be traced together.

VSD: Summary

- Sorting in the right order is key to all of them.
- If expensive lighting/shading is used, do not shade an image pixel more than once.
- For quick rendering, z-buffer algorithms are better.
- BSP Trees can be fast if environment is static.
- Ease of implementation and scope of hardware acceleration are also important.
- Z-buffering is popular due to memory being cheap.