

## 8. Hidden Surface Elimination

- To eliminate edges and surfaces not visible to the viewer
- Two categories:

**Object space methods:** deal with object definitions directly. Order the surfaces so they can be drawn in a particular order to provide correct image.

**Image space methods:** work as part of the projection process to determine relationship between object points on each projector.

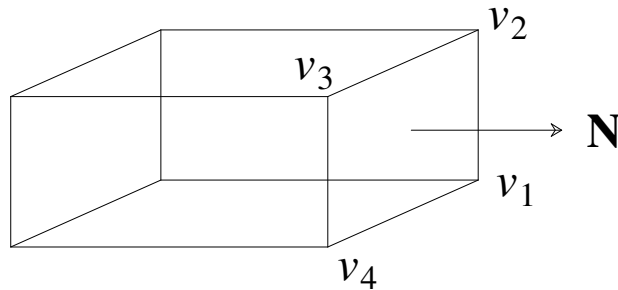
- **Sorting** and **Coherence techniques** are used to improve performance

Sorting: to facilitate depth comparisons

Coherence methods: to take advantages of regularities in a scene

## 8.1 Back Face Removal

- For convex objects, sufficient to remove all **back faces**



1. Compute outward normal:  $\mathbf{N} = (v_2 - v_1) \times (v_4 - v_1)$

2.

(a) Projection type = parallel

a face (polygon) is a back face if  $\mathbf{DOP} \cdot \mathbf{N} > 0$

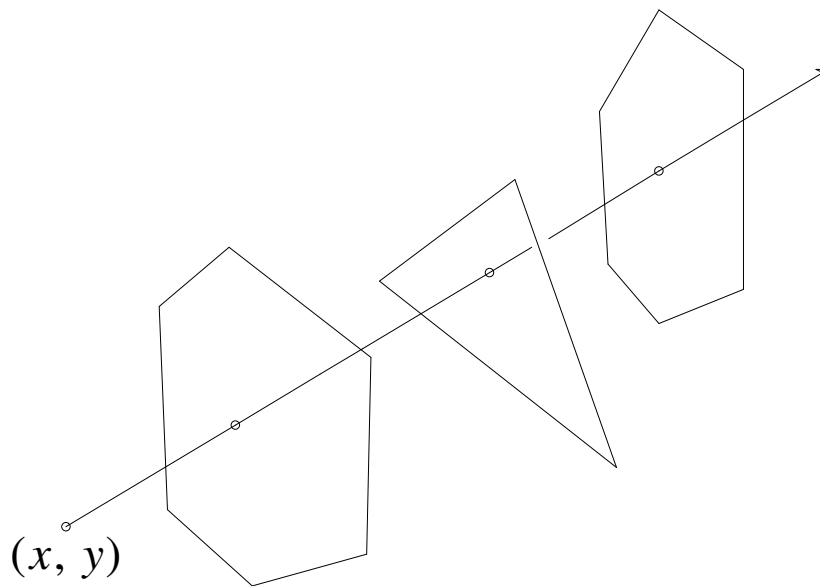
(b) Projection type = perspective

a face (polygon) is a back face if  $\mathbf{V} \cdot \mathbf{N} > 0$

(  $\mathbf{V}$  is a vector from  $\mathbf{COP}$  to any vertex of the face)

## 8.2 Z-Buffer (Depth-Buffer) Method

- Image space method
- Simplest method, one polygon at a time
- Requires two arrays: **Intensity** and **Depth**, indexed by pixel coordinates  $(x, y)$
- For each pixel  $(x, y)$  of the display screen, keep the depth of the object that lies closest to the viewer within the pixel in **Depth** $[x, y]$ , and intensity value at the point of the object in **Intensity** $[x, y]$



## Algorithm:

1. For each pixel  $(x, y)$  of the screen

$\text{Depth}[x, y] \leftarrow -1.0$

$\text{Intensity}[x, y] \leftarrow \text{background intensity (color)}$

2. (Scan Conversion)

For each polygon in the scene, find all the pixels that lie within the boundary of the polygon when projected onto the view plan.

For each of these pixels  $(x, y)$

(2.1)

calculate the depth  $z$  of the polygon at  $(x, y)$

(2.2)

If  $( z > \text{Depth}[x, y] )$  then

(a)  $\text{Depth}[x, y] \leftarrow z$

(b)  $\text{Intensity}[x, y] \leftarrow \text{intensity or shading value of the polygon at } (x, y)$

3. Copy Intensity into the frame (refresh) buffer

## Notes:

- Polygons have to be transformed into (normalized) viewing coordinates and clipped against the normalized view volume first
- Calculation of depth can be done as follows:  
"Record the plane equation of each polygon in the (normalized) viewing coordinate system and then use incremental method to find the depth  $z$ "

$$Ax + By + Cz + D = 0 \quad \text{plane equation}$$

$$\rightarrow z = \frac{-D - Ax - By}{C}$$

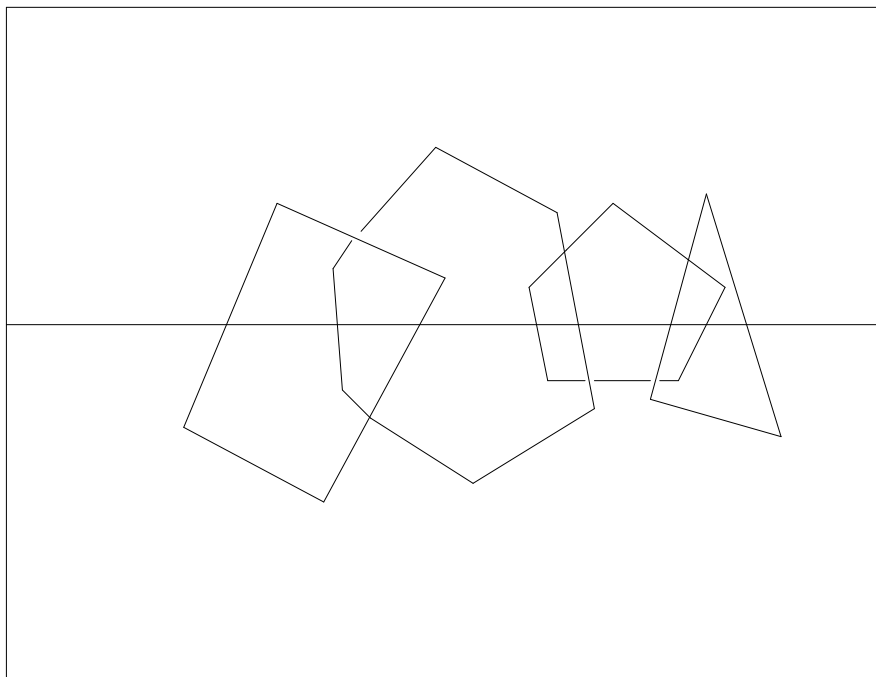
$$\text{Hence, depth at } (x + \Delta x, y) = z - \frac{A}{C} (\Delta x)$$

Only one addition/subtraction is required to compute depth at  $(x + \Delta x, y)$

## 8.3 Scan-Line Method

- An extension of the 2D scan-conversion algorithm
- Image-space method
- Deals with multiple polygons

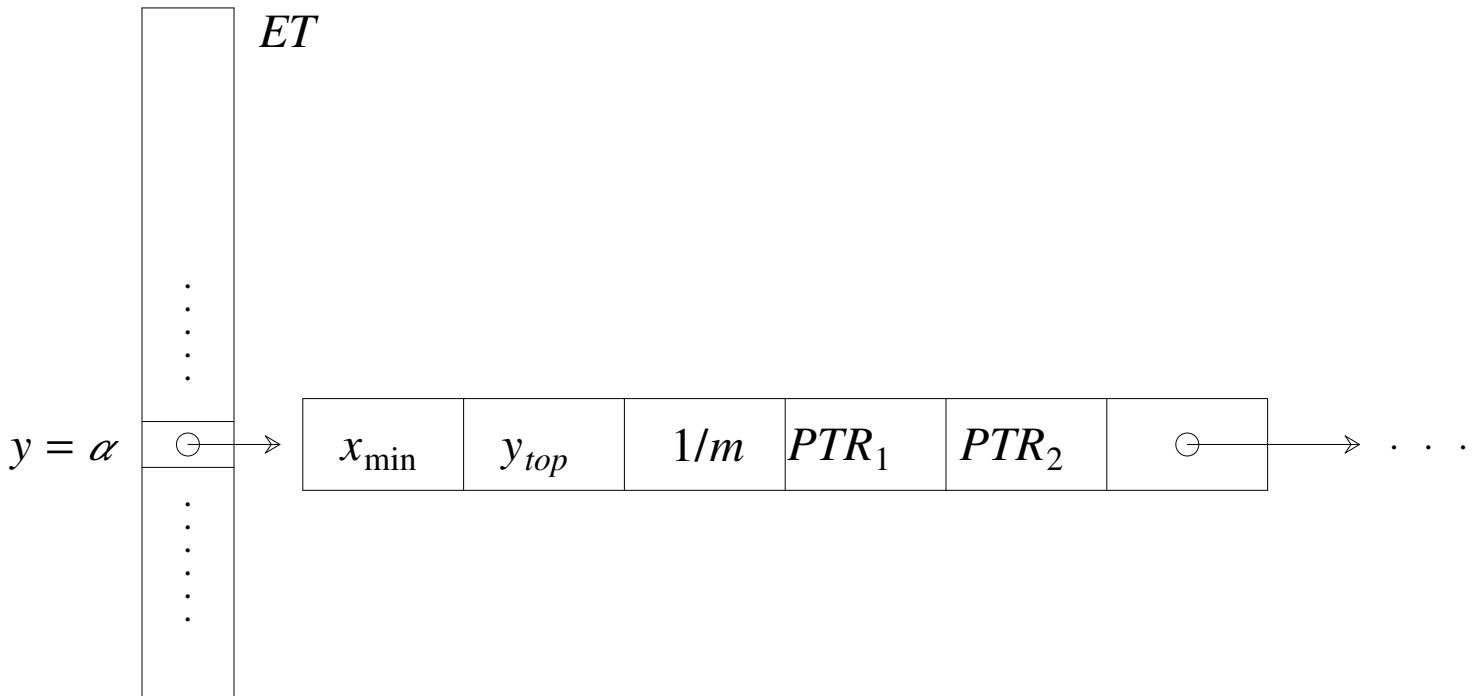
### General idea:



1. Create segments of polygons by intersecting the polygons with the plane represented by a scan line
2. Sort all segments endpoints by  $x$  to identify all the spans of the scan line

3. If no segments appear in a span, the background intensity is used for this span
4. If only one segment is contained in a span then the segment is visible and the polygon equation is used to compute the intensity values for all the pixels in this span
5. If several segments extend accross the entire span then the segment closest to the viewer, the one with the smallest  $z$  value, is found and it's intensity is used for this span.

Need 2 tables: **Bucket Sorted Edge table (ET)**  
**Polygon table (PT)**



- $PTR_1$  and  $PTR_2$  are pointers to the polygons in PT that share this edge

PT

⋮			
No of vertices	Plane equation	Shading (color) information	○
⋮			

To Index  
→  
Table

Need 2 lists:     **Active-edge list (AEL),**  
                    **Active polygon list (APL)**

AEL: edges intersecting current scan line

APL: count, polygons overlapping current span  
(updated for each new span)

## The algorithm

1. Set  $y$  to the smallest  $y$ -coordinate that has a non-empty bucket in ET
2. Set AEL and APL to empty;  $count \leftarrow 0$ .
3. Repeat the following steps while  $y \leq YMAX$

(3.1)

Merge the edges in bucket  $y$  of ET with the edges in AEL in sorted order on  $x$ , and let  $[x_1, x_2]$ ,  $[x_2, x_3]$ , ..., and  $[x_{m-1}, x_m]$  be the corresponding spans on the current scan line.

(3.2)

If AEL is not empty then

for  $i = 1$  to  $m - 1$  do

for each edge in AEL whose  $x_{\min}$  equals  $x_i$ , if the polygon that contains this edge is already in APL then remove this polygon from APL and decrease *count* by 1; otherwise, put this polygon in APL and increase *count* by 1.

if  $count > 0$  then

if  $count = 1$  then compute the intensity values of the pixels between  $x_i$  and  $x_{i+1}$  using the equation of the polygon in APL

else ( $count > 1$ ) compute the intensity values of the pixels between  $x_i$  and  $x_{i+1}$  using the equation of the polygon that is closest to the viewer

else /\*  $count = 0$  \*/

Paint the span with background color

(3.3)

Remove edges in AEL whose  $y_{top}$  equals current  $y$ .

(3.4)

For each edge remaining in AEL, replace  $x_{min}$  with  $x_{min} + 1/m$

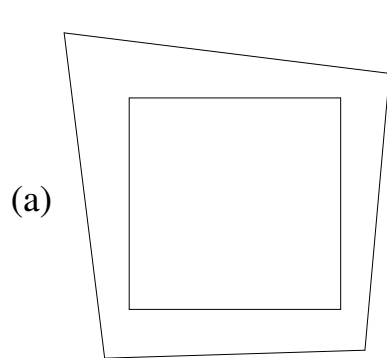
(3.5)

Increment  $y$  by 1 (to the next scan line)

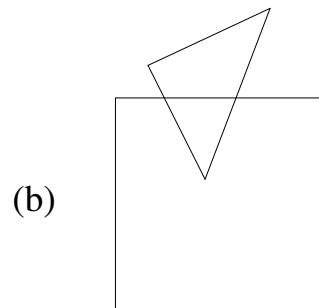
## 8.4 Area-Subdivision Method

- Image-space method
- Use "divide and conquer" and "area coherence"
- Recursively subdivide the projection plane image until a decision can be made

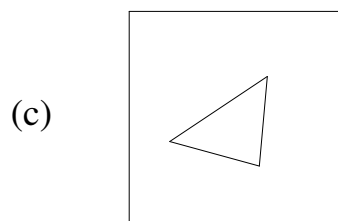
Projection of a polygon has one of four relationships to the area of interest:



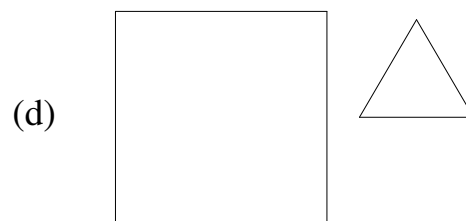
Surrounding polygon



Intersecting polygon



Contained polygon



Disjoint polygon

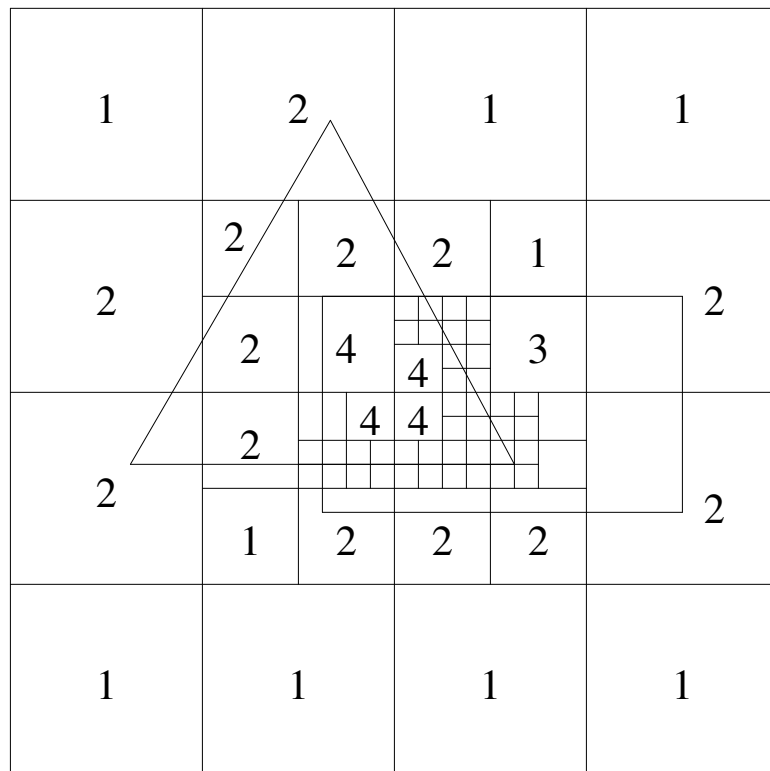
Decision can be made in four cases about an area and, therefore, no further subdivision is required for such area:

1. All polygons are disjoint from the area:

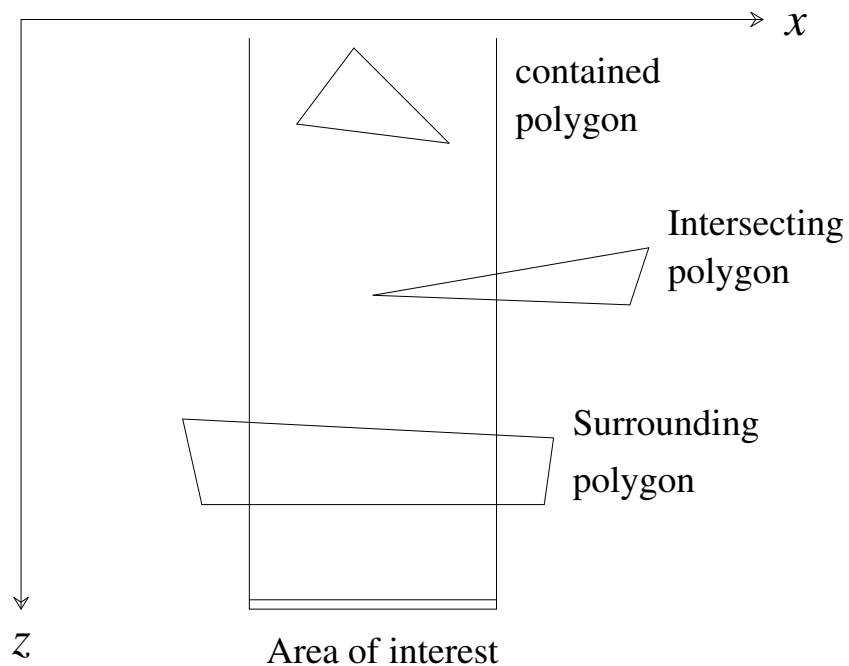
- set intensity of pixels in the area to the background value

2. Only one intersecting or contained polygon:

- fill the area with background intensity value and then scan convert the polygon



3. A single *surrounding* polygon, no intersecting or contained polygon:
  - fill the area with the intensity value of the surrounding polygon
4. More than one polygon is intersecting, contained, or surrounding the area, at least one of them is a *surrounding* polygon:
  - if one of the surrounding polygons is in front of all the other polygons (by testing the z-coordinates at the corners of the area) then display the intensity value of the polygon; otherwise, keep subdividing

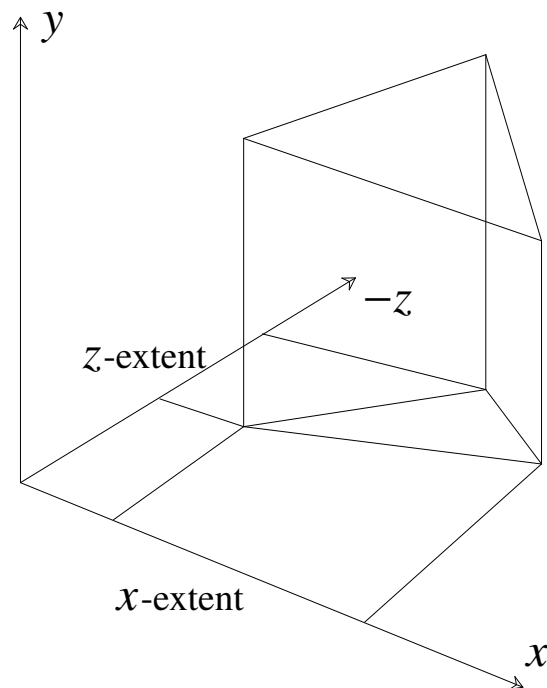


## Notes:

1. After subdivision, only contained and intersecting polygons need to be re-examined.
2. Subdivision ends when the resolution of the display surface has been reached, i.e., when the area is less than that of a pixel
3. Subdivision can be done in several ways:
  - (a) into squares
  - (b) about the polygon vertices
  - (c) along polygon boundaries

## 8.5 Depth-Sort (Priority) Method

- Objec-space method
- Polygons are sorted according to their distance from the viewer (depth) and then scan-converted in reversed order, i.e., the farthest polygon is displayed first, then the second farthest polygon, ..., and finally, the closest polygon.
- Each polygon requires  $x$ -,  $y$ -, and  $z$ -extents in the data structure.



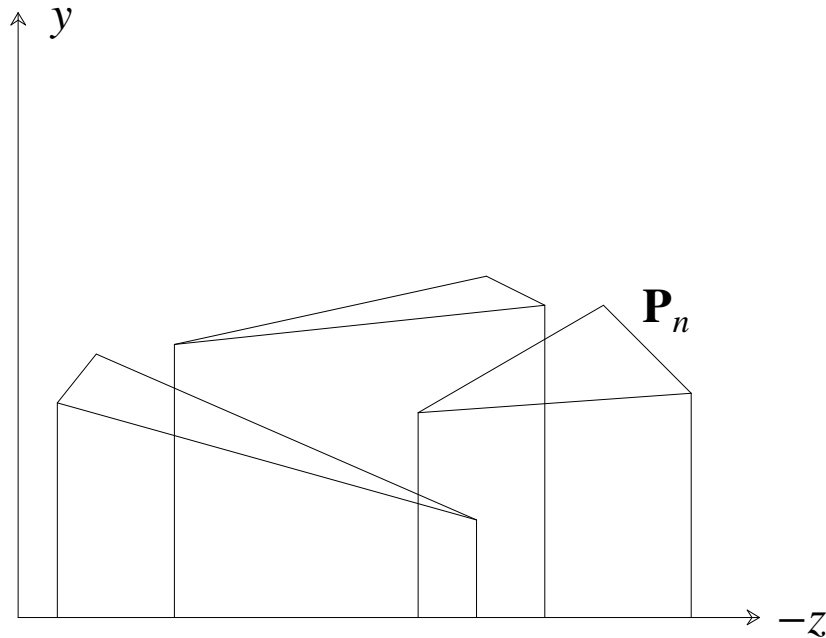
## Hidden Surface Removal:

1. Sort all polygons  $\mathbf{P}_i$  according to the smallest  $z$ -coordinates of their vertices into a list in descending order

$$\mathbf{P} : \mathbf{P}_1 \rightarrow \mathbf{P}_2 \rightarrow \mathbf{P}_3 \rightarrow \cdots \rightarrow \mathbf{P}_n$$

2. Resolve ambiguities by doing the following tests starting at the end of  $\mathbf{P}$  and scan-convert polygons in ascending order. Polygons in  $\mathbf{P}$  are not marked initially.

2.1 If  $n = 1$  then scan-convert  $\mathbf{P}_1$  and halt.



2.2 Sort all polygons  $\mathbf{P}_i$  ( $1 \leq i \leq n - 1$ ) whose  $z$ -extents overlap that of  $\mathbf{P}_n$  into a list

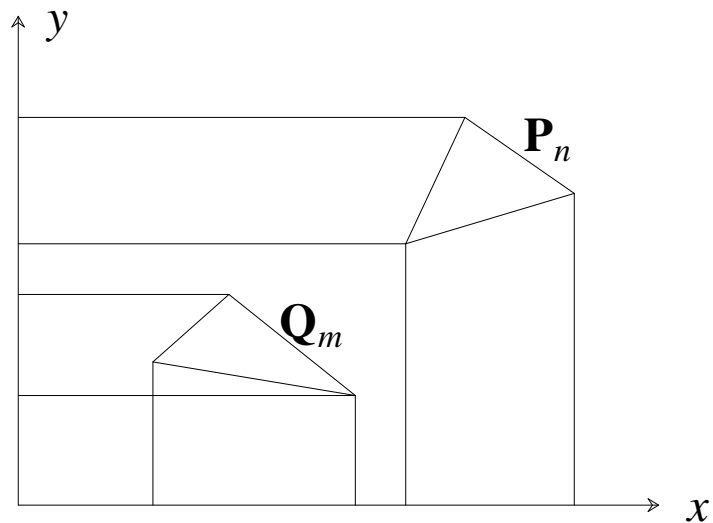
$$\mathbf{Q} : \mathbf{Q}_1 \rightarrow \mathbf{Q}_2 \rightarrow \mathbf{Q}_3 \rightarrow \cdots \rightarrow \mathbf{Q}_m$$

in the same order as the polygon list  $\mathbf{P}$ .

2.3 If  $\mathbf{Q}$  is empty then scan-convert  $\mathbf{P}_n$ ,  $n \leftarrow n - 1$ , and goto Step 2.1

2.4 If the  $x$ -extents of  $\mathbf{P}_n$  and  $\mathbf{Q}_m$  do not overlap then goto Step 2.11

2.5 If the  $y$ -extents of  $\mathbf{P}_n$  and  $\mathbf{Q}_m$  do not overlap then goto Step 2.11

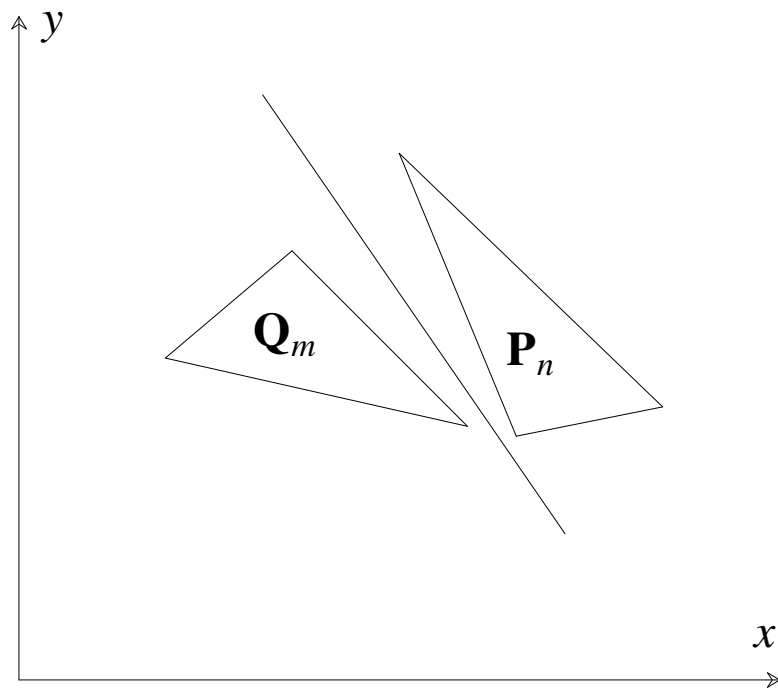


2.6 If  $\mathbf{P}_n$  is on different side of  $\mathbf{Q}_m$  from the viewer then goto Step 2.11

2.7 If  $\mathbf{Q}_m$  is on the same side of  $\mathbf{P}_n$  as the viewer then goto Step 2.11

2.8 If the projections of  $\mathbf{P}_n$  and  $\mathbf{Q}_m$  onto the  $xy$ -plane do not overlap then go to Step 2.11

2.9 If  $\mathbf{P}_n$  is not marked then swap  $\mathbf{P}_n$  and  $\mathbf{Q}_m$  in the sorted list  $\mathbf{P}$ , mark  $\mathbf{P}_n$  and goto Step 2.11



2.10 If  $\mathbf{P}_n$  is marked then

- (i) bisect  $\mathbf{P}_n$  by the plane of  $\mathbf{Q}_m$  into two polygons  $\mathbf{P}'_n$  and  $\mathbf{P}''_n$
- (ii) delete  $\mathbf{P}_n$  from  $\mathbf{P}$
- (iii) insert  $\mathbf{P}'_n$  and  $\mathbf{P}''_n$  into  $\mathbf{P}$
- (iv)  $n \leftarrow n + 1$  and goto Step 2.2

2.11 If  $m = 1$  then scan-convert  $\mathbf{P}_n$ ,  $n \leftarrow n - 1$ , and goto Step 2.1. Otherwise,  $m \leftarrow m - 1$  and goto Step 2.4.

**Note:** Some polygons will be un-necessarily scan-converted.

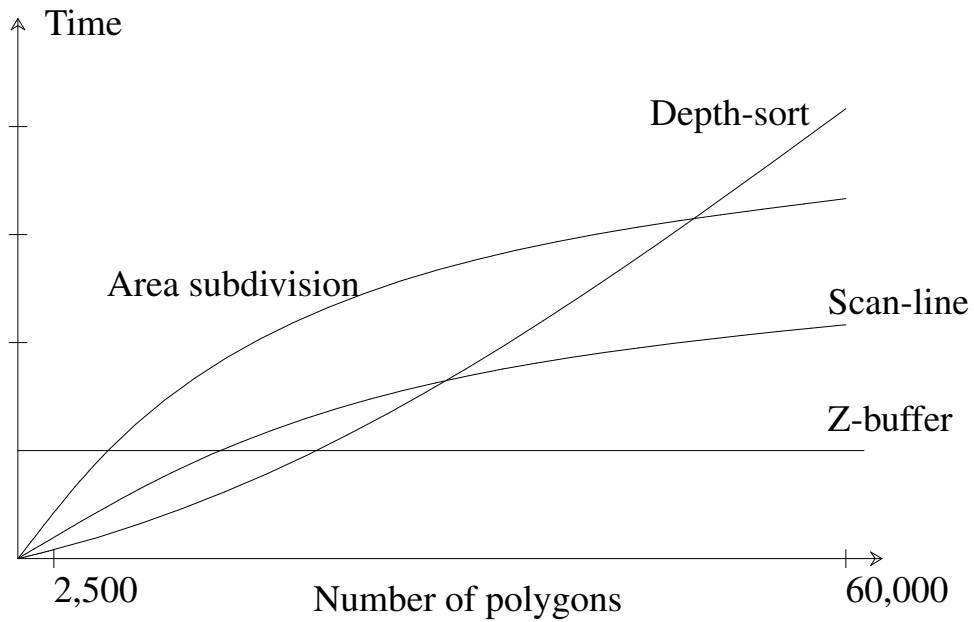
### **Hidden Line Removal:**

Use the same algorithm except that entries of the refresh buffer should be set to some value  $v_0$  (background value) first. When a polygon is scan-converted, its edges are set to a different value  $v_1$  and its interior pixels are set to  $v_0$ .

## Algorithm Efficiency

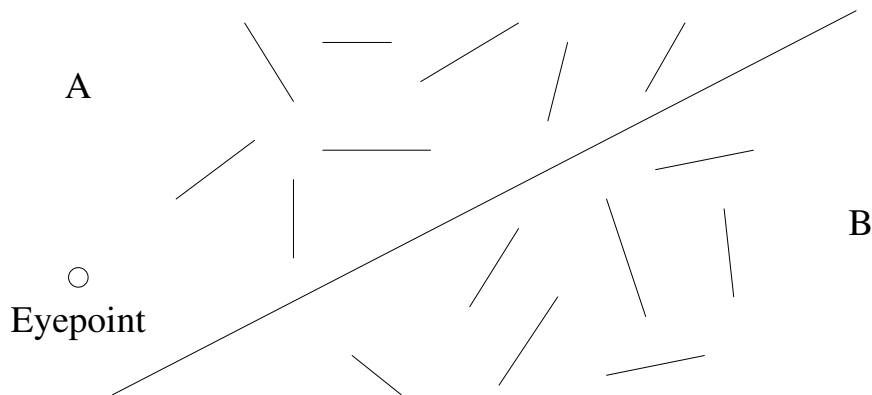
Number of Polygons	100	2,500	60,000
Depth sort	1*	10	507
Z-buffer	54	54	54
Scan-line	5	21	100
Area subdivision	11	64	107

\* Entries normalized



## 8.6 Binary Space-Partition (BSP) Trees (Fuchs, Kedem, Naylor)

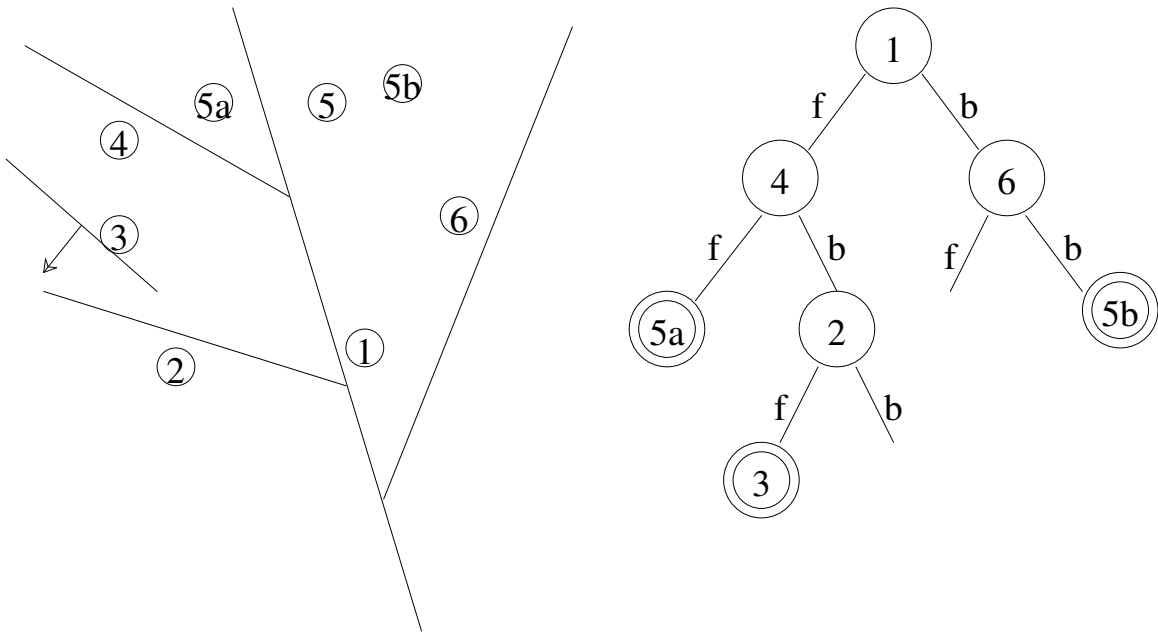
- Efficient for calculating visibility relationships among 3D polygons (from any view point)
- Based on the following concept



A polygon on the same side of the plane as the eyepoint can not be obscured by polygons on the other side.

Given a set of 3D polygons (with assigned normal directions), a **BSP tree** can be constructed as follows:

*"Choose an arbitrary polygon as the root polygon. Use the root polygon to partition the environment into two half spaces: **front** and **back** (relative to polygon normal). Any polygon lying on both sides of the root polygon's plane is split. Then choose an arbitrary polygon on each side to divide the remaining polygons in its half-space in the same fashion. This process is recursively repeated until each region contains at most one polygon."*



## How to use a BSP tree to calculate visibility?

For a given view point, recursively display polygons of the tree in the following order:

- if the view point is in the root polygon's front half-face, display polygons in the root's rear half-space, the root polygon and then polygons in its front half-space.
- if the view point is in the root polygon's rear half-space, display polygons in reverse order
- if the view point is on the plane that contains the root polygon then either way is okay.

