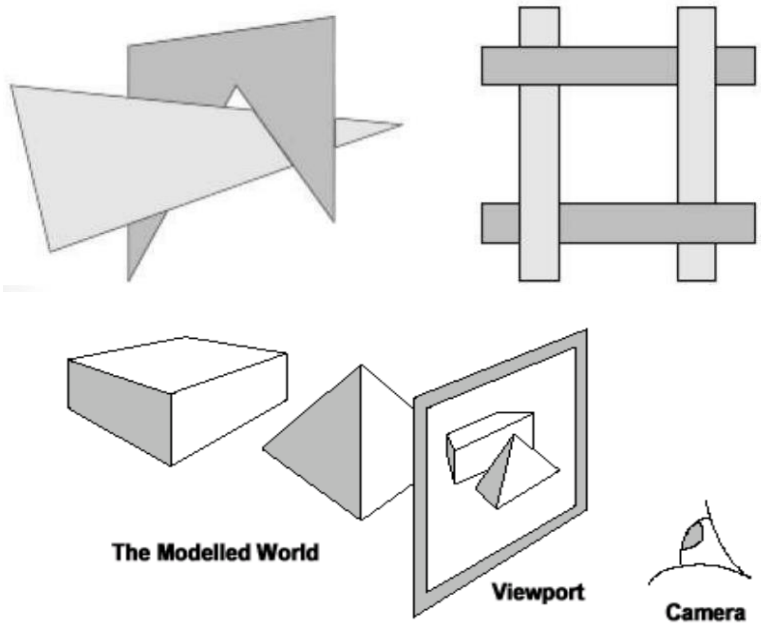
Chapter 7 Visible Surface Determination

It is the process of identifying those parts of a scene that are visible from a chosen viewing position. There are numerous algorithms for efficient identification of visible objects for different types of applications. These various algorithms are referred to as visible-surface detection methods. Sometimes these methods are also referred to as hidden-surface elimination methods.

To identify those parts of a scene that are visible from a chosen viewing position (visible-surface detection methods).

Surfaces which are obscured by other opaque (solid) surfaces along the line of sight are invisible to the viewer so can be eliminated (hidden-surface elimination methods).



Visible surface detection methods are broadly classified according to whether they deal with objects or with their projected images.

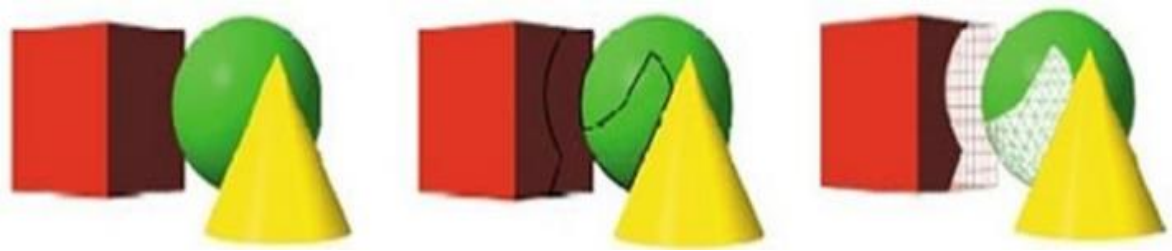
Object-Space Methods (OSM):

- Algorithm to determine which parts of the shapes are to be rendered in 3D coordinates.
- Methods based on comparison of objects for their 3D positions and dimensions with respect to a viewing position
- For N objects, may require N*N comparison operations.
- Efficient for small number of objects but difficult to implement.
- Depth sorting, area subdivision methods.

- Deal with object definitions directly
- Compares objects and parts of objects to each other within the scene definition to determine which surface as a whole we should label as visible.
- It is continuous method.
- Compares each object with all other objects to determine the visibility of the object parts.
- E.g. Back-face detection method

Image-Space Methods(ISM):

- Deal with projected images of the objects and not directly with objects.
- Visibility is decided point by point at each pixel position on the projection plane.
- It is a discrete method.
- Accuracy of the calculation is bounded by the display resolution.
- A change of display resolution requires re-calculation.
- Based on the pixels to be drawn on 2D. Try to determine which object should contribute to that pixel.
- Running on the pixels to be drawn on 2D. Try to determine which object should contribute to that pixel.
- E.g. Depth-buffer method, Scan-line method, Area-subdivision method



Back Face Detection:

- A fast and simple object-space method for identifying the back faces of a polyhedron.
- It is based on the performing inside-outside test

Chapter 7 Visible Surface Determination

Methods:

1st Method:

- A point (x, y, z) is "inside" a polygon surface with plane parameters A, B, C, and D if $Ax + By + Cz + D < 0$ (from plane equation).
- When an inside point is along the line of sight to the surface, the polygon must be a back face. (We are inside that face and cannot see the front of it from our viewing position)
- In eq. $Ax + By + Cz + D = 0$
  if A,B,C remain constant , then varying value of D result in a whole family of parallel plane

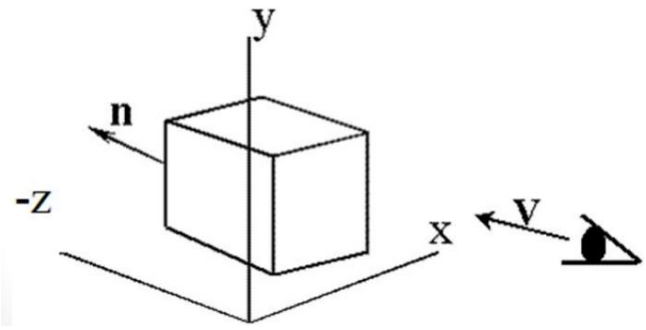
  if D>0, plane is behind the origin (Away from observer) the surface is invisible.

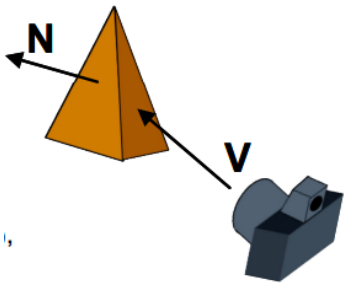
  if D<=0 , plane is in front of origin (toward the observer) the surface is visible.

2nd Method:

- To simplify this test, Let N be normal vector to a polygon surface, which has Cartesian components (A, B, C). In general, if V is a vector in the viewing direction from the eye (or "camera") position, then this polygon is a back face

if **V.N>0.**





If object description have been converted to projection coordinates and our viewing direction is parallel to the viewing z_v axis, then $\vec{V} = (0, 0, V_z)$ and

$$\vec{V}.\vec{N} = VzC$$

so that we only need to consider the sign of C, the z component of the normal vector $\vec{N}$.

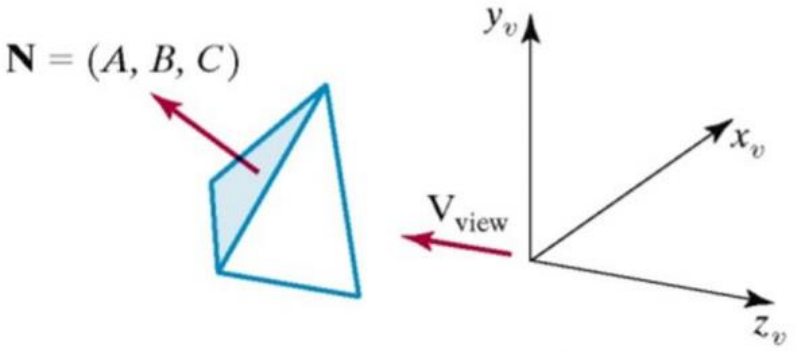


Figure. A polygon surface with plane parameter C<0 in a right-handed viewing coordinate system is identified as a back face when the viewing direction is along the negative z_v axis.

In a right-handed viewing system with viewing direction along the negative z_v axis, the polygon is a backface if C<0. Also, we cannot see any face whose normal has z component C = 0, since our viewing direction is grazing that polygon. Thus, in general, we can label any polygon as a backface if its normal vector has a z component value

$$C \leq 0.$$

Q. Find the visibility for the surface AED in rectangular pyramid where an observer is at P (5, 5, 5).

Solution

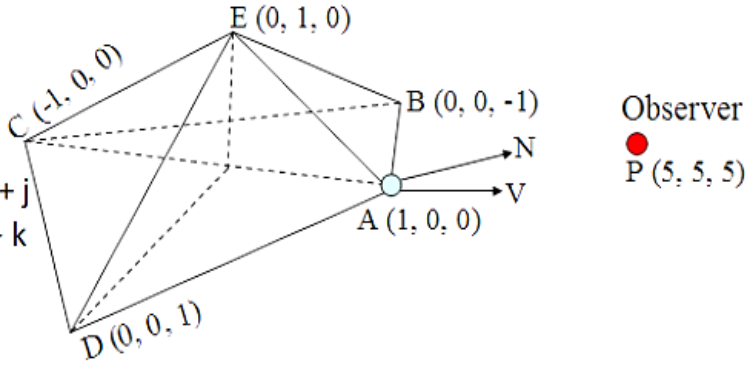
Here,

AE = (0-1)i + (1-0)j + (0-0)k = -i + j

AD= (0-1)i + (1-0)j + (1-0)k =-i + k

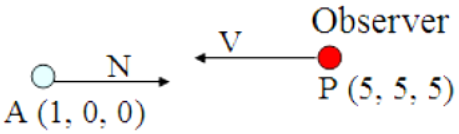
Step-1:

Normal vector N for AED



Thus, N = AE x AD = $\begin{pmatrix} i & j & k \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$ $= i(1-0) - j(-1+0) + k(0+1)$
$= i + j + k$

Case-I



Step-2: If observer at P(5, 5, 5) so we can construct the view vector

V from surface to view point A(1, 0, 0) as:

V = PA = (1-5)i + (0-5)j + (0-5)k = -4i - 5j - 5k

Step-3: To find the visibility of the object, we use dot product of view vector V and normal vector N

as:

V.N = (-4i - 5j - 5k).(i + j + k) = -4-5-5 = -14< 0

This shows that the surface is visible for the observer.

Case-II

Step-2: If observer at P(5, 5, 5) so we can construct the view vector V from surface to view point A(1, 0, 0) as:

V = AP = (5-1)i + (5-0)j + (5-0)k = 4i + 5j + 5k

Step-3: To find the visibility of the object, we use dot product of view vector V and normal vector N as:

V.N = (4i + 5j + 5k).(i + j + k) = 4+5+5 = 14> 0

This shows that the surface is invisible for the observer.

Q. Find the visibility for the surface AED in rectangular pyramid where an observer is at P (0, 0.5, 0).

Chapter 7 Visible Surface Determination

Doesn't works well for

1. Overlapping front faces due to
   - Multiple objects
   - Concave objects
2. Non Closed Objects
3. Non-polygonal models

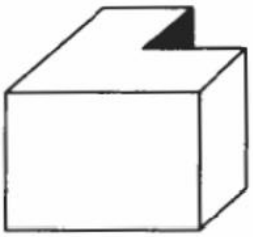


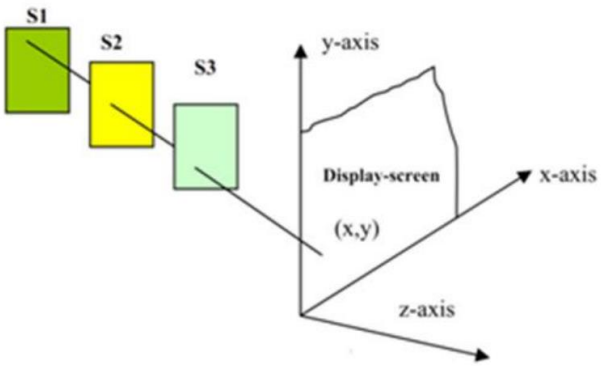
Fig: View of a concave polyhedron with one face partially hidden by other faces

Depth –Buffer Method (Z –Buffer Method):

- A commonly used image-space approach to detecting visible surfaces is the depth-buffer method, which compares surface depths at each pixel position on the projection plane.
- Also called z-buffer method since depth usually measured along z-axis. This approach compares surface depths at each pixel position on the projection plane.
- Each surface of a scene is processed separately, one point at a time across the surface. And each (x, y, z) position on a polygon surface corresponds to the projection point (x, y) on the view plane.

This method requires two buffers:

- A **z-buffer** or **depth buffer**: Stores depth values for each pixel position (x, y).
- **Frame buffer** (**Refresh buffer**): Stores the surface-intensity values or color values for each pixel position.
- As surfaces are processed, the image buffer is used to store the color values of each pixel position and the z-buffer is used to store the depth values for each (x, y) position.



Chapter 7 Visible Surface Determination

Initially, all positions in the depth buffer are set to 0 (minimum depth), and the refresh buffer is initialized to the background intensity. Each surface listed in the polygon tables is then processed, one scan line at a time, calculating the depth (z-value) at each (x, y) pixel position. The calculated depth is compared to the value previously stored in the depth buffer at that position. If the calculated depth is greater than the value stored in the depth buffer, the new depth value is stored, and the surface intensity at that position is determined and placed in the same xy location in the refresh buffer.

A drawback of the depth-buffer method is that it can only find one visible surface for opaque surfaces and cannot accumulate intensity values for transparent surfaces.

Algorithm:

1. Initialize both, depth buffer and refresh buffer for all buffer positions (x, y),
   depth(x, y) = 0
   refresh(x, y) = $I_{background}$,
   (where $I_{background}$ is the value for the background intensity.)

2. Process each polygon surface in a scene one at a time,
   (Each surface listed in the polygon tables is then processed, one scan line at a time, calculating the depth (z-value) at each (x, y) pixel position.)
   2.1. Calculate the depth z for each (x, y) position on the polygon.
   (The calculated depth is compared to the value previously stored in the depth buffer at that position.)
   2.2. If Z > depth(x, y), then set
   depth(x, y)=z (If the calculated depth is greater than the value stored in the depth buffer, the new depth value is stored,)
   refresh(x, y)= I_{surf}(x, y),
   (where I_{surf}(x, y) is the intensity value for the surface at pixel position (x, y).)

3. After all pixels and surfaces are compared, draw object using X,Y,Z from depth and intensity refresh buffer.

After all surfaces have been processed the depth buffer contains depth values for the visible surfaces and the refresh buffer contains the corresponding intensity values for those surfaces

Depth value for a surface position (x, y) is

$z = (-Ax - By - D)/c$ .....(i)

Let depth z' at (x + 1 , y)

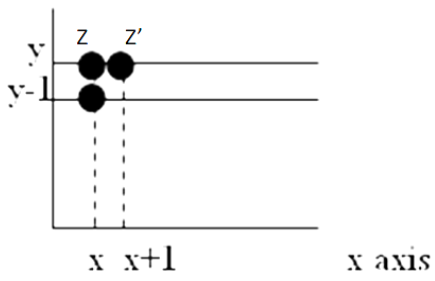
$z' = \{-A(x+1) - By - D\}/c$

$z' = \{-Ax - By - D-A\}/c$

$z' = (-Ax - By - D)/c - A/c$

or

$z' = z - A/c$.....(ii)



The ratio –A/C is constant for each surface, so succeeding depth values across a scan line are obtained from proceeding values with a single addition.

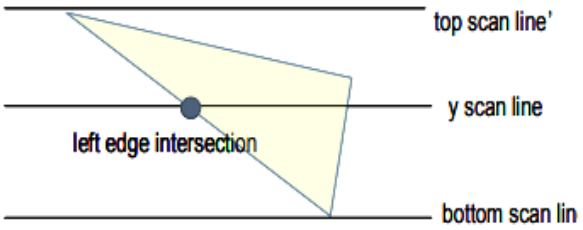
On each scan line, we start by calculating the depth on a left edge of the polygon that intersects that scan line. Depth values at each successive position across the scan line are then calculated by equation (ii).

Start from top scan line to bottom scan line

Starting from top vertex, calculate x position down the left edge of the polygon recursively as

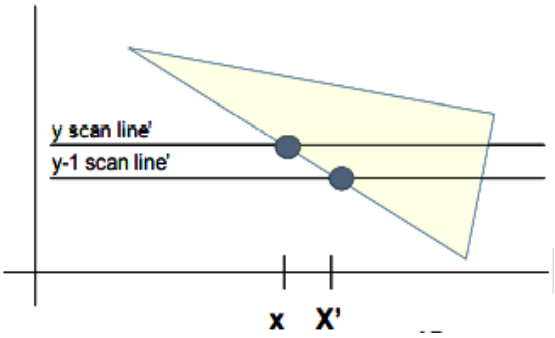
$x' = x - 1/m$

Thus depth value for the next pixel in left edge is obtained as



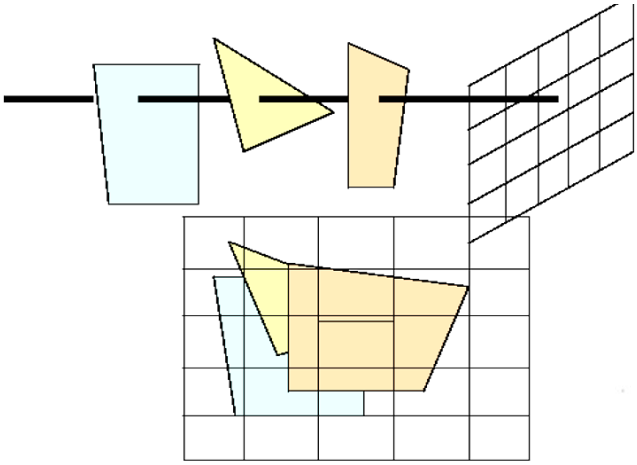
$$z' = z + \frac{\frac{A}{m} + B}{C}$$

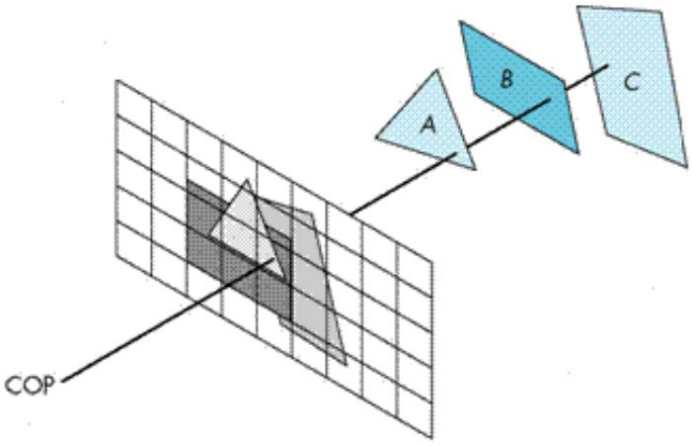
put x' = x-1/m and y' = y-1



For vertical edge m = infinity, so:

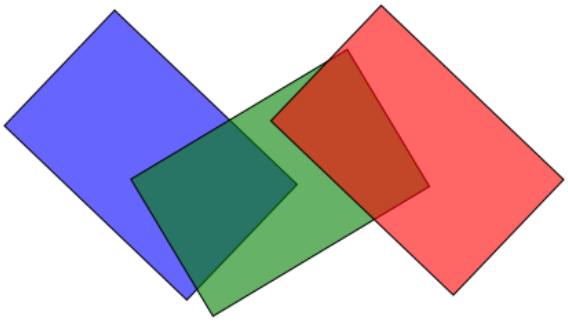
$$z' = z + \frac{B}{C}$$





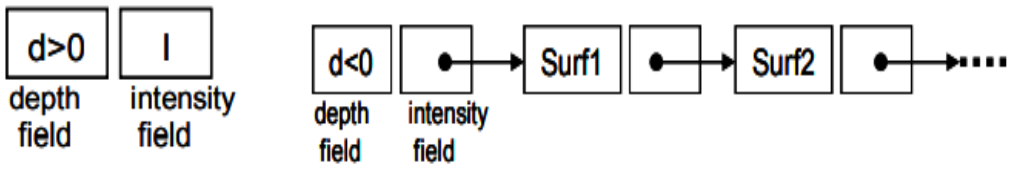
A-Buffer Method:

- The A-buffer (**anti-aliased**, **area-averaged**, **accumulation buffer**) is an extension of the ideas in the depth-buffer method (other end of the alphabet from "z-buffer").
- A drawback of the depth-buffer method is that it deals only with opaque (Solid) surfaces and cannot accumulate intensity values for more than one transparent surfaces.
- The A-buffer method is an extension of the depth-buffer method so that each position in the buffer can reference a linked list of surfaces. Thus, more than one surface intensity can be taken into consideration at each pixel position, and object edges can be anti-aliased.



Each position in the A-buffer has two fields:

- **Depth Field:** Stores a positive or negative real number
  - **Positive: Single** surface contribute to pixel intensity
  - **Negative**: **Multiple** surfaces contribute to pixel intensity
- **Intensity Field:** Stores surface-intensity information or a pointer value.
  - **Surface intensity** if **single** surface stores the RGB components of the surface color at that point
  - and percent of pixel coverage **Pointer value** if **multiple** surfaces



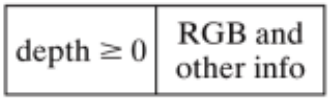
Data for each surface in the linked list includes

- RGB intensity components
- Opacity parameter (percent of transparency)
- Depth
- Percent of area coverage
- Surface identifier
- Other surface-rendering parameters
- Pointer to next surface

If depth is >= 0, then the surface data field stores the depth of that pixel position as before (SINGLE SURFACE)

If the depth field is positive, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area.

The intensity field then stores the RCB components of the surface color at that point and the percent of pixel coverage, as in figure.

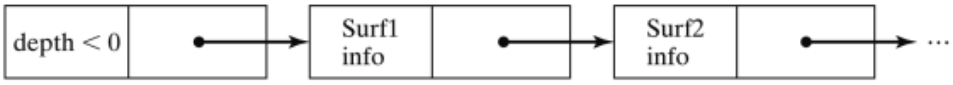


If depth < 0 then the data filed stores a pointer to a linked list of surface data (MULTIPLE SURFACE)

If the depth field is negative, this indicates multiple-surface contributions to the pixel intensity.

The intensity field then stores a pointer to a linked list of surface data, as in figure.

Data for each surface in the linked list includes: RGB intensity components, opacity parameter (percent of transparency), depth, and percent of area coverage, surface identifier, other surface-rendering parameters, and pointer to next surface.



- Scan lines are processed to determine surface overlaps of pixels across the individual scan lines.
- Surfaces are subdivided into a polygon mesh and clipped against the pixel boundaries
- The opacity factors and percent of surface overlaps are used to determine the pixel intensity as an average of the contribution from the overlapping surfaces

Chapter 7 Visible Surface Determination

Scan-Line method

- This image-space method for removing hidden surfaces is an extension of the scan-line algorithm for filling polygon interiors where, we deal with multiple surfaces rather than one.
- Each scan line is processed with calculating the depth for nearest view for determining the visible surface of intersecting polygon. When the visible surface has been determined, the intensity value for that position is entered into the refresh buffer.
- To facilitate the search for surfaces crossing a given scan line, we can set up an active list of edges from information in the edge table that contain only edges that cross the current scan line, sorted in order of increasing x.
- In addition, we define a flag for each surface that is set on or off to indicate whether a position along a scan line is inside or outside of the surface. Scan lines are processed from left to right.
- At the leftmost boundary of a surface, the surface flag is turned on; and at the rightmost boundary, it is turned off.

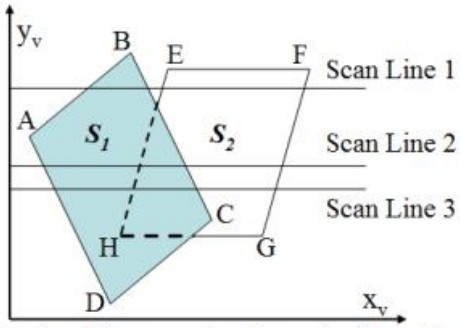


Fig. Scan lines crossing the projection of two surfaces, S_1 and S_2 in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

Data Structure:

A. Edge table containing
- Coordinate endpoints for each line in a scene
- Inverse slope of each line
- Pointers into polygon table to identify the surfaces bounded by each line

B. Surface table containing
- Coefficients of the plane equation for each surface
- Intensity information for each surface
- Pointers to edge table

C. Active Edge List
- To keep a trace of which edges are intersected by the given scan line

Note :
- The edges are sorted in order of increasing x
- Define flags for each surface to indicate whether a position is inside or outside the surface

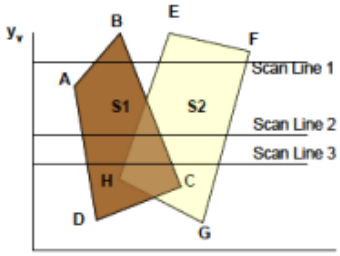
Chapter 7 Visible Surface Determination

I. Initialize the necessary data structure

1. Edge table containing end point coordinates, inverse slope and polygon pointer.
2. Surface table containing plane coefficients and surface intensity
3. Active Edge List
4. Flag for each surface

II. For each scan line repeat

1. update active edge list
2. determine point of intersection and set surface on or off.
3. If flag is on, store its value in the refresh buffer
4. If more than one surface is on, do depth sorting and store the intensity of surface nearest to view plane in the refresh buffer



For scan line 1

- The active edge list contains edges AB,BC,EH, FG
- Between edges AB and BC, only *flags for s1 == on* and between edges EH and FG, only *flags for s2==on*
  - no depth calculation needed and corresponding surface intensities are entered in refresh buffer

For scan line 2

- The active edge list contains edges AD,EH,BC and FG
- Between edges AD and EH, only the *flag for surface s1 == on*
- Between edges EH and BC *flags for both surfaces == on*
  - Depth calculation (using plane coefficients) is needed.
- In this example ,say s2 is nearer to the view plane than s1, so intensities for surface s2 are loaded into the refresh buffer until boundary BC is encountered
- Between edges BC and FG flag for s1==off and *flag for s2 == on*
- Intensities for s2 are loaded on refresh buffer

For scan line 3

- Same **coherent** property as scan line 2 as noticed from active list, so no depth

Chapter 7 Visible Surface Determination

Problem:

Dealing with **cut through** surfaces and **cyclic overlap** is problematic when used coherent properties

- Solution: Divide the surface to eliminate the overlap or cut through