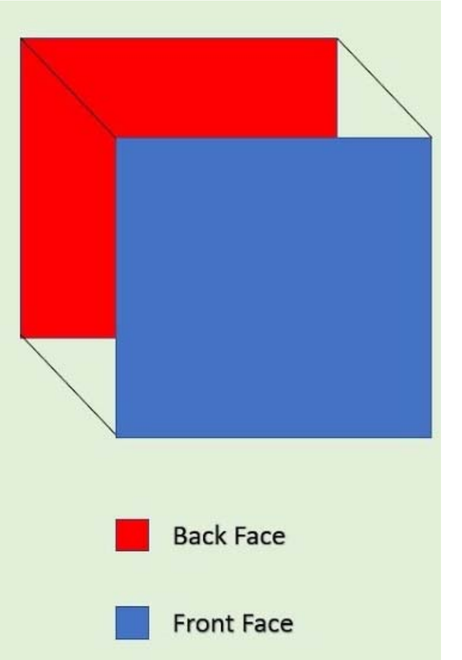
Chapter 5

Visible Surface Detection

Hidden surface elimination/visible surface detection

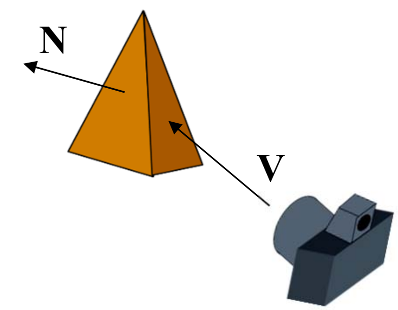
- A procedure that removes any such surfaces or line which are not to be displayed in a 3D scene.
- A large number of algorithms exists.
- Some algorithms:
    - Require More memory
    - Require More processing time
    - Apply to special types of objects only
- Deciding upon which methods to use depends on many factors like:
    - Complexity of the scene, types of objects to be displayed, available equipment, whether static or animated displays etc.



Hidden surface elimination/visible surface detection

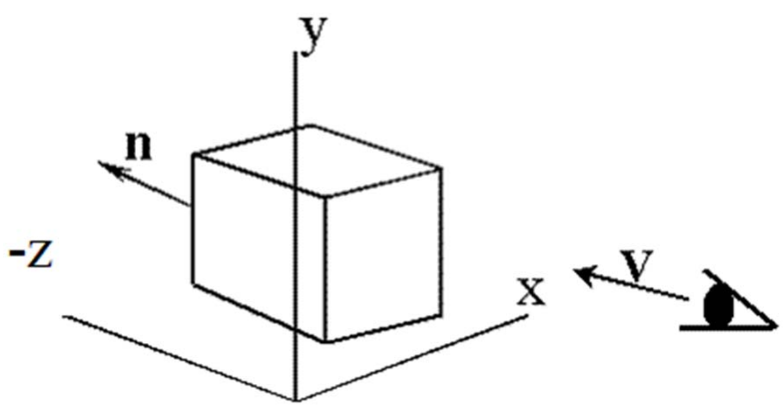
- Broadly classified Two approaches
  - Depends upon object definition or projected image
    - Object-space method
      - Compares objects and parts of objects to each other within the scene
    - Image-space method
      - Visibility is decided point by point at each pixel position on projected plane
      - This method is by far the common

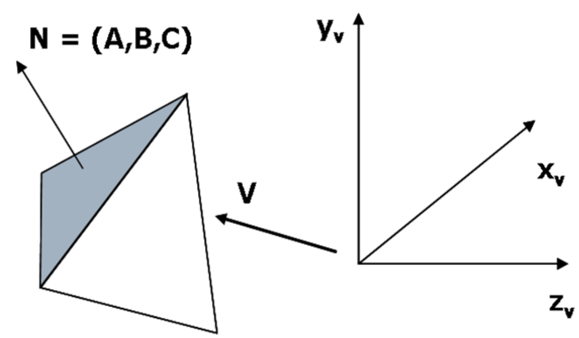
Back Face Detection Method



- Fast and simple object-space method
- Based on “inside” or “outside” test
- Test whether view point is inside or outside the polygon surface
  - Tested by the inequality: Ax + By + Cz + D = 0
    - If view point is outside the surface → visible
    - If inside → back face
  - The test could be simplified by considering normal vector **N** = (A,B,C) to polygon surface and Vector **V** in viewing direction as:

    the polygon is back face if: **V.N**>0





Back Face Detection Method

- If object description is converted to viewing co-ordinates then, **V** is along z_v axis i.e **V** = $(0,0,V_z)$, then:

  $\mathbf{V.N} = V_z C$

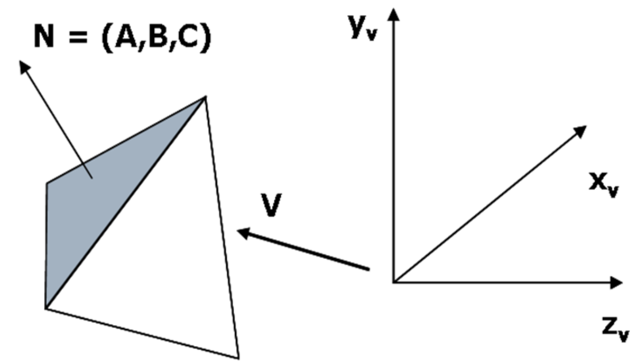
  So, $V_z C > 0 \rightarrow$

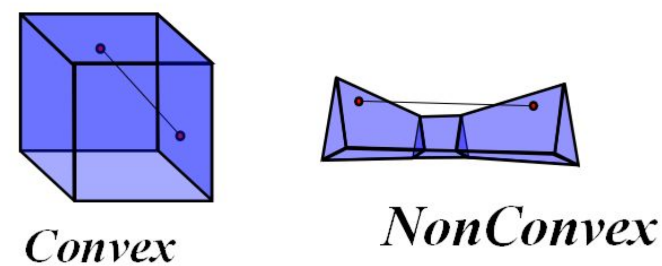
  So we need to consider only sign of C
- For right handed viewing system with viewing direction along negative z_v axis,

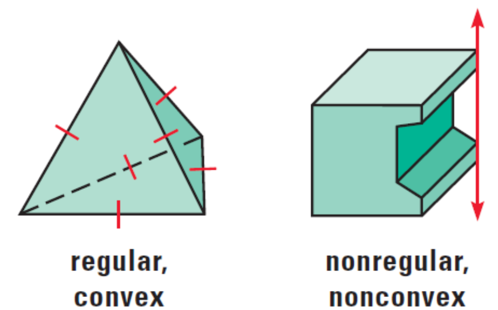
  if $c<0 \rightarrow$ back face

  What if $C = 0$??

  - Grazing Viewing i.e viewing direction is perpendicular to surface normal→ also back face. So, back face if $C \leq 0$
- Convex polyhedron?→ no problem because either completely visible or completely hidden
- Concave polyhedron?→ some problem; needs additional faces
- More than one objects? in scene → problem







Depth Buffer (z-Buffer) method

- Image-space method
- Surface depth is compared at each pixel position on the projection plane
- Usually applied to scene containing only polygon surfaces; but possible to apply for non-planar surfaces
- For each pixel position (x,y) on projection plane, object depths can be compared by comparing z-values as each (x,y,z) position on a polygon surface corresponds to the orthographic projection point (x,y) on projection plane
- Two buffers required
  - Depth buffer: to store depth value for each position
  - Refresh buffer: to store intensity values of each position
- Drawback: Deals only with opaque surface but not with transparent surface

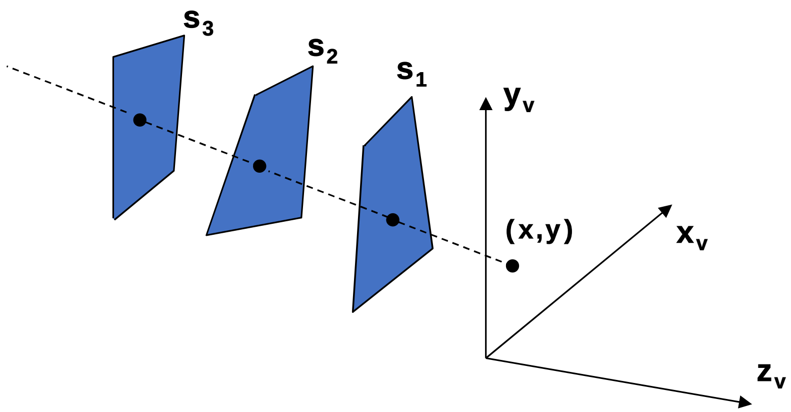
Depth Buffer (z-Buffer) method

1. Initialize the depth buffer and refresh buffer so that for all buffer positions (x,y),

   depth(x,y) = 0 (min Value), refresh(x,y) = $I_{backgrnd}$

2. For each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility
   - Calculate the depth z for each (x,y) position on the polygon
   - If z>depth(x,y), then set

     depth(x,y)=z, refresh(x,y)=I_{surf}(x,y)

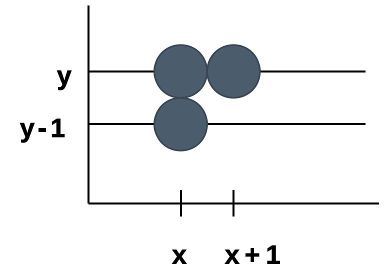


$I_{backgrnd}$ = background intensity
I_{surf}(x,y) = projected intensity value for the surface at pixel position (x,y)

Depth Buffer (z-Buffer) method

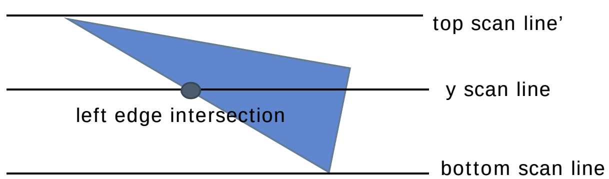
- From plane equation, depth is

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



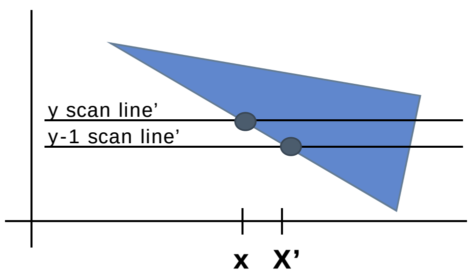
- For the next adjacent pixel in a scan line, depth is

$$z' = \frac{-A(x+1) - By - D}{C} \quad \text{or} \quad z' = z - \frac{A}{C}$$



- Determine minimum and maximum y-coordinates for each polygon
- 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   m(x'-x)=y'-y=-1



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

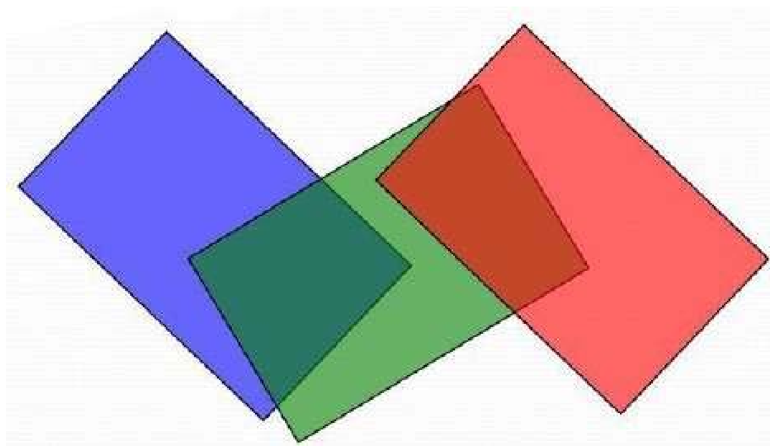
$$z' = z + \frac{\frac{A}{m} + B}{C}$$

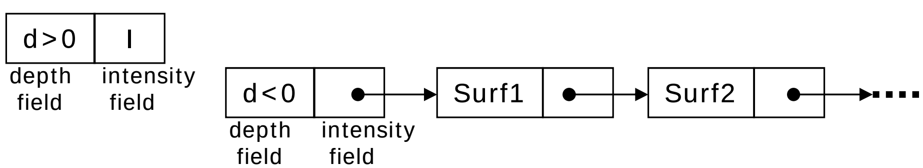
put x' = x-1/m and y' = y-1

- For vertical edge m = infinity, so: $z' = z + \frac{B}{C}$

A-Buffer Method

- Extension of depth buffer method
- Antialiased, area-average, accumulation-buffer
- The depth buffer is expanded so that each position in the buffer can reference a linked list of surfaces thus enabling more than one surface intensity consideration
- Each pixel position in the A-Buffer has two fields
  - Depth Field → stores a positive or negative real number
    - Positive→ single surface contributes 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





A-Buffer Method

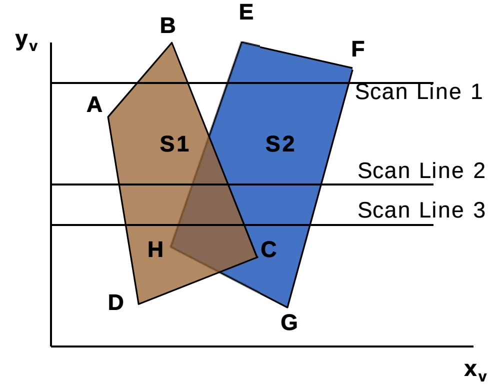
- in case of surface linked list, the data for each surface in the linked list includes
  - RGB intensity
  - Opacity parameter
  - Depth
  - Percent of area coverage
  - Surface identifier
  - Other surface-rendering parameters
  - Pointer to next surface
- Scanlines are processed to determine surface overlaps of pixels across the individual scanlines.
- 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

Scan-Line Method

- Image space method
- Deals with multiple surfaces
- Construct edge tables and polygon tables
  - Edge table contains → endpoints of each edge in the scene, inverse slope of each line and pointers to the polygon table to identify its corresponding surface
  - Polygon table contains → plane equation coefficients, surface intensity, pointers to the edge table (optional)
- Setup an active list (why active ??) of edges for those edges which cross the current scan line
  - 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
  - At leftmost boundary of surface flag == *on* and at rightmost boundary flag == *off*

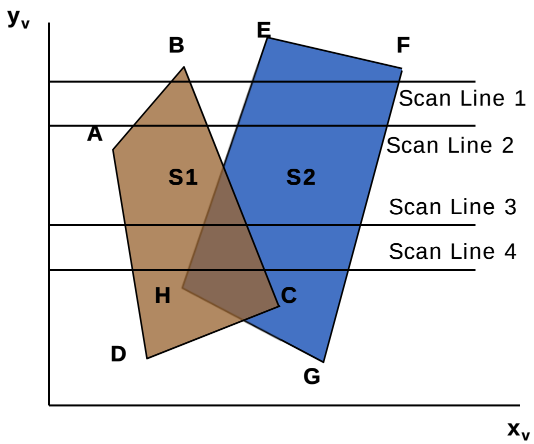
Scan-Line Method

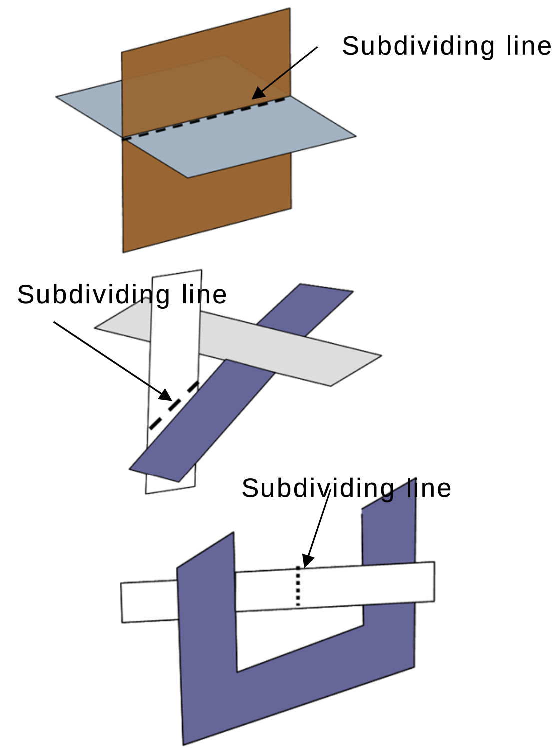
- For scan line 1
  - The active edge list contains edges AB,BC,EH, FG
  - Between edges AB and BC, only *flags for s1 == on* and between edges EH and FG, only *flags for s2==on*
    - no depth calculation needed and corresponding surface intensities are entered in refresh buffer
- For scan line 2
  - 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 depth for s1 is less than for s2, so intensities for surface s1 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 calculation needed between edges BC and EH



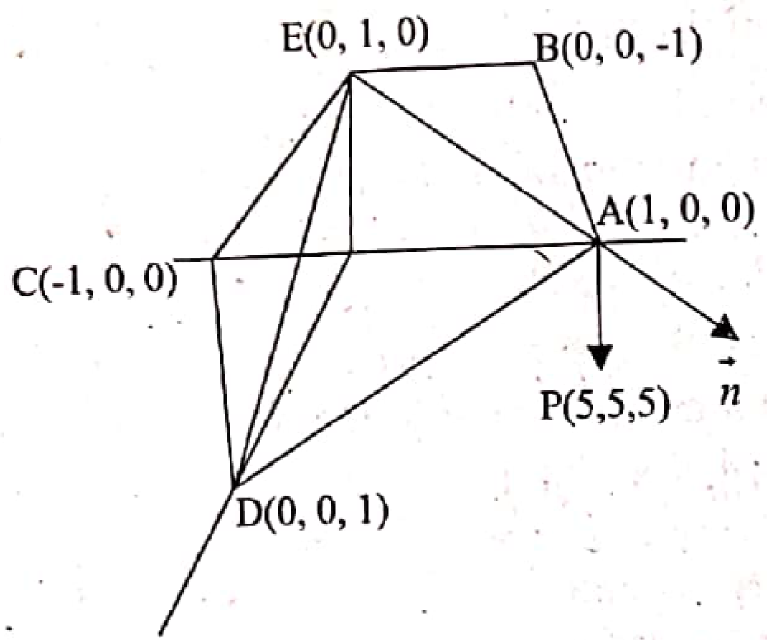
Scan-Line Method

- 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





Find the visibility for the surface AED when the observer is at P(5, 5, 5)



Determine whether two surfaces of an object with normal $2\vec{i} - 3\vec{j} + 4\vec{k}$ and $\vec{i} + \vec{j} - 2\vec{k}$ respectively, viewed from the direction given by $\vec{i} - \vec{j} + \vec{k}$ are backface or frontface?