Introduction to Visible Surface Detection Methods:

In the realistic graphics display, we have to identify those parts of a scene that are visible from a chosen viewing position. The various algorithms that are used for that are referred to as **Visible-surface detection methods** or **hidden-surface elimination** methods.

Types of Visible Surface Detection Methods:

- Object-space methods and
- Image-space methods

Visible-surface detection algorithms are broadly classified according to whether they deal with object definitions directly or with their projected images. These two approaches are called object-space methods and image-space methods, respectively.

An **object-space method** compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible. In an **image-space algorithm**, visibility is decided point by point at each pixel position on the projection plane. Most visible-surface algorithms use image-space methods, although object space methods can be used effectively to locate visible surfaces in some cases. Line display algorithms, on the other hand, generally use object-space methods to identify visible lines in wire frame displays, but many image-space visible-surface algorithms can be adapted easily to visible-line detection.

Visible Surface Detection Methods:

We will see four methods for Detecting Visible surface Methods. They are:

1. Back Face Detection Method
2. Depth Buffer Method
3. Scan line Method
4. Depth Sorting Method

1. Back Face Detection Method:

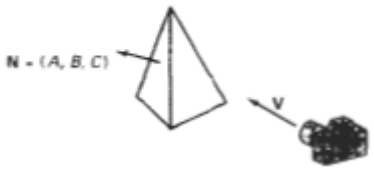
1. A fast and simple object-space method for identifying the back faces of a polyhedron is based on the "inside-outside" tests. A point *(x,* y, z) is "inside" a polygon surface with plane parameters A, B, C, and D if

$$Ax + By + Cz + D < 0$$

2. 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).
3. We can simplify this test by considering the normal vector N 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, as shown in Fig, then this polygon is a back face if

V.N>0

$C \leq 0$



4. Furthermore, if object descriptions have been converted to projection coordinates and our viewing direction is parallel to the viewing z,. axis, then V = (0, 0, *Vz)* and

V.N=Vz.C

so that we only need to consider the sign of C, the ; component of the normal vector N .

5. In a right-handed viewing system with viewing direction along the negative z, axis, the polygon is a back face 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 back face if its normal vector has a z-component value:

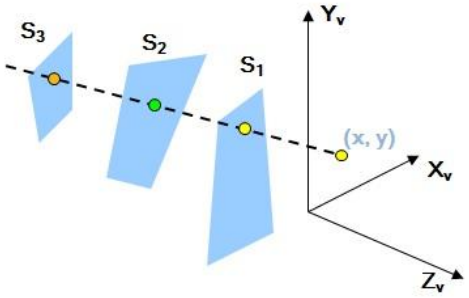
$$C \leq 0$$

6. In a Left-handed viewing system with viewing direction along the positive z, axis, the polygon is a back face 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 back face if its normal vector has a z-component value:

$$C \geq 0$$

2. Depth Buffer Method(also called as Z-Buffer method):

1. 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. This procedure is also referred to as the z-buffer method, since object depth is usually measured from the view plane along the z axis of a viewing system.
2. Each surface of a scene is processed separately, one point at a time across the surface. The method is usually applied to scenes containing only polygon surfaces, because depth values can be computed very quickly and the method is easy to implement. But the method can be applied to non planar surfaces.
3. With object descriptions converted to projection coordinates, each *(x,* y, z*)* position on a polygon surface corresponds to the orthographic projection point *(x,* y) on the view plane. Therefore, for each pixel position *(x,* y) on the view plane, object depths can be compared by comparing z values.
4. Below Figure shows three surfaces at varying distances along the orthographic projection line from position *(x ,*y) in a view plane taken as the *xy, plane*. Surface 1, is closest at this position, so its surface intensity value at (x, y) is saved. As implied by the name of this method, two buffer areas are required.



5. Here two buffer's are required:
   a. Depth Buffer(store's depth value for each pixel)
   b. Refresh Buffer(store's intensity values for each pixel)
6. 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.
7. The calculated depth is compared to the value previously stored in the depth buffer at that position. If the calculated depth is patter than the value stored in the depth buffer, the new depth value is stored, and the surface intensity at that position is determined and in the same *xy* location in the refresh buffer.

Steps in Depth Buffer Algorithm:

1. Initialize the depth buffer and refresh buffer so that for all buffer positions (x,y),
   depth(x,y) = 0, refresh(x,y)= $I_{background}$ (background Intensity)
2. For each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility.
   a) Calculate the depth z for each (x,y) position on the polygon.
   b) If z>depth(x,y), then set
      Depth(x,y)=z, refresh(x,y)= I_{surf}(x,y)

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.

Calculation of Depth Values when processing from left to right direction:

- Depth values (z values) are calculated from plane equation: **Ax+By+Cz+D=0**
- Depth value for surface position (x, y) is Z=$(-Ax - By - D)/C$
- Depth value for surface position (x+1, y) is Z'=$(-A(x+1) - By - D)/C$ or z'=z-(A/C)

Calculation of Depth Values when processing from top to bottom vertex:

We first determine the y-coordinate extents of each polygon, and process the surface from the topmost scan line to the bottom scan line.

Starting at a top vertex, we can recursively calculate x positions down a left edge of the polygon as **x' = *x* - (1/m)**, where *m* is the slope of the edge.

Depth values down the edge are then obtained recursively as

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

If we are processing down a vertical edge, the slope is infinite and the recursive calculations reduce to

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

3. SCANLINE METHOD:

1. This method is extension of the scan-line algorithm for filling polygon interiors
2. This method is an example of image space method.
3. For all polygons intersecting each scan line
   - Processed from left to right
   - Depth calculations for each overlapping surface
   - The intensity of the nearest position is entered into the refresh buffer

polygon tables:

The following polygon tables are used to store coordinate descriptions of polygons along with surfaces.

Vertex Table: contains all vertices and their coordinates

Edge table : contains all edge names and their coordinate endpoints

Surface facet table: contains all surfaces along with their corresponding edge names.

Edge table

- Coordinate endpoints for each line
- Slope of each line
- Pointers into the polygon table
  - Identify the surfaces bounded by each line

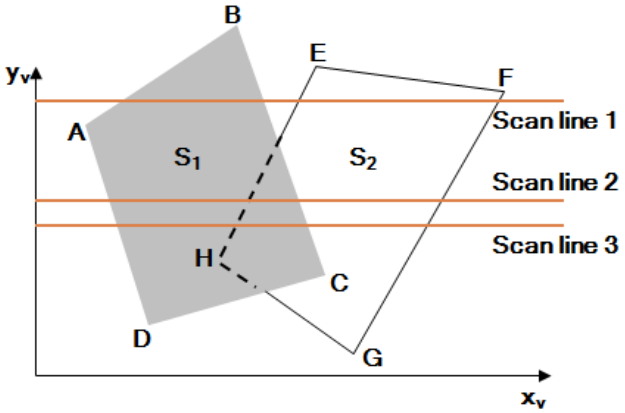
Surface table

- Coefficients of the plane equation for each surface
- Intensity information for the surfaces
- Pointers into the edge table

4. For each scanline, maintain the following things: **Active Edge table and Surface Flag**
   - **Active edge list**
     - Contain only edges across the current scan line
     - Sorted in order of increasing x
   - **Flag for each surface**
     - Indicate whether inside or outside of the surface
     - At the leftmost boundary of a surface
       - The surface flag is turned on
     - At the rightmost boundary of a surface
       - The surface flag is turned off

Example:

- Active list for scan line 1
  - Edge table
    - AB, BC, EH, and FG
    - Between AB and BC, only the flag for surface S_1 is on
    - No depth calculations are necessary
    - Intensity for surface S_1 is entered into the refresh buffer
    - Similarly, between EH and FG, only the flag for S_2 is on
  - For scan line 2, 3
    - AD, EH, BC, and FG
      - Between AD and EH, only the flag for S_1 is on
      - Between EH and BC, the flags for both surfaces are on
        - Depth calculation is needed
        - Intensities for S_1 are loaded into the refresh buffer until BC



- Take advantage of coherence
  - Pass from one scan line to next
  - Scan line 3 has the same active list as scan line 2
  - Unnecessary to make depth calculations between EH and BC

4. DEPTH SORTING ALGORITHM (also known as Painters Algorithm):

This algorithm involves both object space and image space operations.

- **Image-space and object-space operations**
  - Sorting operations in both image and object-space
  - The scan conversion of polygon surfaces in image-space
- **Basic functions**
  - Surfaces are sorted in order of decreasing depth
  - Surfaces are scan-converted in order, starting with the surface of greatest depth

1. This algorithm also referred to as the ***painter's algorithm***

   In creating an oil painting
   - First paints the background colors
   - The most distant objects are added
   - Then the nearer objects, and so forth
   - Finally, the foregrounds are painted over all objects
   - Each layer of paint covers up the previous layer

   this algorithm process the surfaces in the above order only.
2. **Process**

   Sort surfaces according to their distance from the view plane

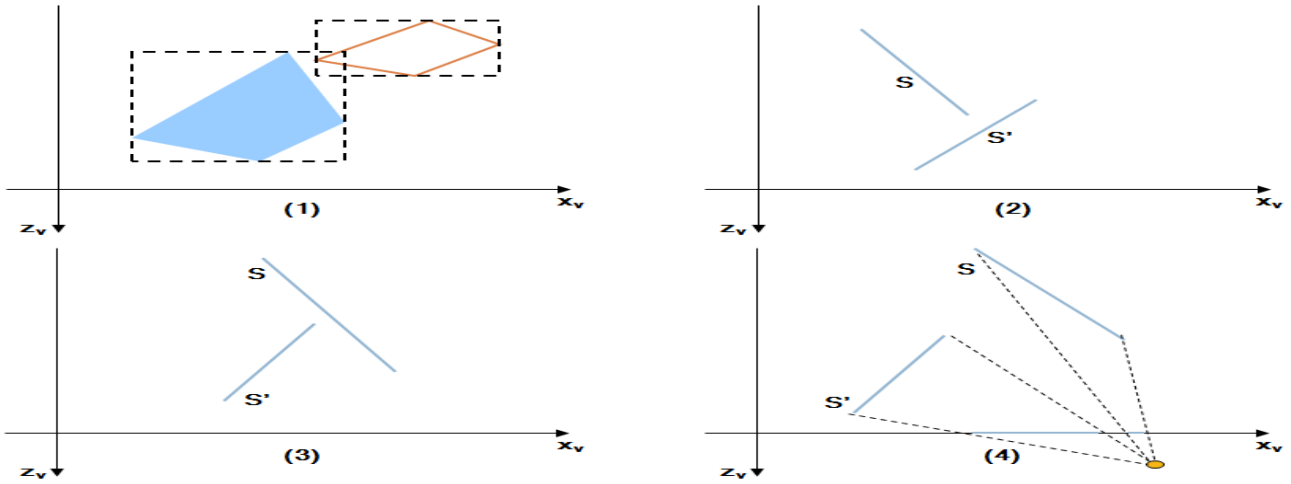
   The intensities for the farthest surface are then entered into the refresh buffer

   Taking each succeeding surface in decreasing depth order
3. Sorting the surfaces can be done only by taking the following tests into consideration

   S-surface which is being tested

   S'-overlapping surface
   - Tests for each surface that overlaps with S
     - The bounding rectangle in the xy plane for the two surfaces do not overlap (1)
     - Surface S is completely behind the overlapping surface(S') relative to the viewing position (2)
     - The overlapping surface(S') is completely in front of S relative to the viewing position (3)
     - The projections of the two surfaces onto the view plane do not overlap (4)
   - If all the surfaces pass at least one of the tests, none of them is behind S
     - No reordering is then necessary and S is scan converted

Overlapping Test Examples



- If all four tests fail with S'
  - Interchange surfaces S and S' in the sorted list
  - Repeat the tests for each surface that is reordered in the list