COMPUTER GRAPHICS

UNIT-3

Two-dimensional Viewing:

The two-dimensional viewing is a transformation process of real-world object into position point which is relative to the viewing volume, especially, the points behind the viewer.

Clipping:

Clipping is a computer graphics process to remove the lines, objects, or line segments, all of which are outside the viewing pane.

2D Viewing Pipeline:

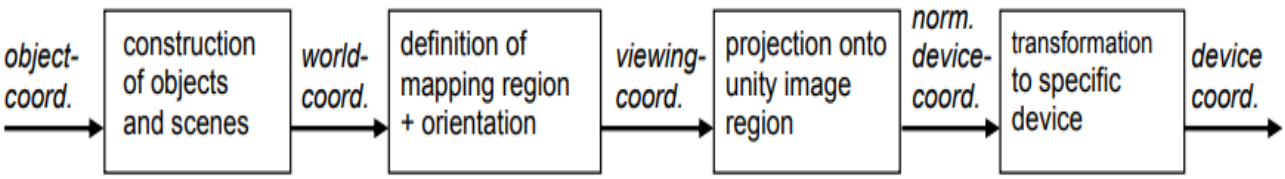
The Term Viewing Pipeline describes a series of transformations, which are passed by geometry data to end up as image data being displayed on a device.

The viewing transformation which maps picture co-ordinates in the WCS to display co-ordinates in PDCS is performed by the following transformations.

- Converting world co-ordinates to viewing co-ordinates.
- Normalizing viewing co-ordinates.
- Converting normalized viewing co-ordinates to device co-ordinates.

The steps involved in viewing pipeline:-

1. Construct the scene in world co-ordinate using the output primitives and attributes.
2. Obtain a particular orientation for the window by setting a two-dimensional viewing co-ordinate system in the world co-ordinate plane and define a window in the viewing co-ordinate system.
3. Use viewing co-ordinates reference frame to provide a method for setting up arbitrary orientations for rectangular windows.
4. Once the viewing reference frame is established, transform descriptions in world co-ordinates to viewing co-ordinates.
5. Define a view port in normalized co-ordinates and map the viewing co-ordinates description of the scene to normalized co-ordinates.
6. Clip all the parts of the picture which lie outside the viewport.



- The coordinates in which individual objects (models) are created are called **model (or object) coordinates.**
- When several objects are assembled into a scene, they are described by **world coordinates.**
- After transformation into the coordinate system of the camera (viewer) they become **viewing coordinates.**
- Their projection onto a common plane (window) yield device-independent **normalized coordinate.**
- Finally, after mapping those normalized coordinates to a specific device, we get **device coordinates.**

Window and viewport:

Capturing images from the real world and displaying them on the screen is an astonishing process, only if we do not know the underlying process. Here, we will be studying how the images are captured. This process is held by the Window port and Viewport in Computer Graphics.

Window:

The window port can be confused with the computer window but it isn't the same. The window port is the area chosen from the real world for display. This window port decides what portion of the real world should be captured and be displayed on the screen. The widow port can thus be defined as,

"A world-coordinate area selected for display is called a window. A window defines a rectangular area in the world coordinates."

Viewport:

Now, the Viewport is the area on a display device to which a window is mapped. Thus, the viewport is nothing else but our device's screen. The viewport can thus be defined as follows:

"A viewport is a polygon viewing region in computer graphics. The viewport is an area expressed in rendering-device-specific coordinates, e.g., pixels for screen coordinates, in which the objects of interest are going to be rendered."

- So, to display the image on the computer screen, we must map our window port to the viewport. The capture ratio of the window might not always be similar to or easily adjustable to the viewport. Thus, some necessary transformations ad adjustments like clipping and cropping are performed on the window.
- The process of mapping the window port to the viewport is termed as 'Window to viewport Transformation'. It is also known as **"Viewing Transformation"** or **"Windowing Transformation"**. It is defined as follows:
- "Window to Viewport Transformation is the process of transforming a 2D world-coordinate object (Window Port) to device coordinates (Viewport). So, objects inside the world or clipping window are mapped to the viewport which is the area on the screen where world coordinates are mapped to be displayed."

Difference between Window Port and Viewport:

Window Port	**Viewport**
Window port is the coordinate area specially selected for the display.	Viewport is the display area of viewport in which the window is perfectly mapped.
Region Created according to World Coordinates.	Region Created according to Device Coordinates.
It is a region selected form the real world. It is a graphically control thing and composed of visual areas along with some of its program controlled with help of window decoration.	It is the region in computer graphics which is a polygon viewing region.
A window port can be defined with the help of a GWINDOW statement.	A viewport is defined by the GPORT command.

Line Clipping (Cohen–Sutherland Algorithm):

In this algorithm, we are given 9 regions on the screen. Out of which one region is of the window and the rest 8 regions are around it given by 4-digit binary. The division of the regions are based on (x_max, y_max) and (x_min, y_min).

The central part is the viewing region or window, all the lines which lie within this region are completely visible. A region code is always assigned to the endpoints of the given line.To check whether the line is visible or not.

Formula to check binary digits: - TBRL which can be defined as top, bottom, right, and left accordingly.

Algorithm:

Steps:

1. Assign the region codes to both endpoints.
2. Perform OR operation on both of these endpoints.
3. if OR = 0000,

then it is completely visible (inside the window).

else

Perform AND operation on both these endpoints.

i) if AND ≠ 0000,

then the line is invisible and not inside the window. Also, it can't be considered for clipping.

ii) else

AND = 0000, the line is partially inside the window and considered for clipping.

4. After confirming that the line is partially inside the window, then we find the intersection with the boundary of the window. By using the following formula:-

Slope:- m= (y2-y1)/(x2-x1)

a) If the line passes through top or the line intersects with the top boundary of the window.

x = x + (y_wmax – y)/m

y = y_wmax

b) If the line passes through the bottom or the line intersects with the bottom boundary of the window.

x = x + (y_wmin – y)/m

y = y_wmin

c) If the line passes through the left region or the line intersects with the left boundary of the window.

y = y+ (x_wmin – x)*m

x = x_wmin

d) If the line passes through the right region or the line intersects with the right boundary of the window.

$$y = y + (x\_wmax - x)*m$$

$$x = x\_wmax$$

5. Now, overwrite the endpoints with a new one and update it.

6. Repeat the 4th step till your line doesn't get completely clipped

- The Cohen–Sutherland algorithm can be used only on a rectangular clip window. For other convex polygon clipping windows, Cyrus–Beck algorithm is used. We will be discussing Cyrus–Beck Algorithm in next set.

Line Clipping (Cyrus Beck Algorithm)

Cyrus Beck is a line clipping algorithm that is made for convex polygons. It allows line clipping for non-rectangular windows, unlike Cohen Sutherland or Nicholl Le Nicholl. It also removes the repeated clipping needed in Cohen Sutherland.

Algorithm:

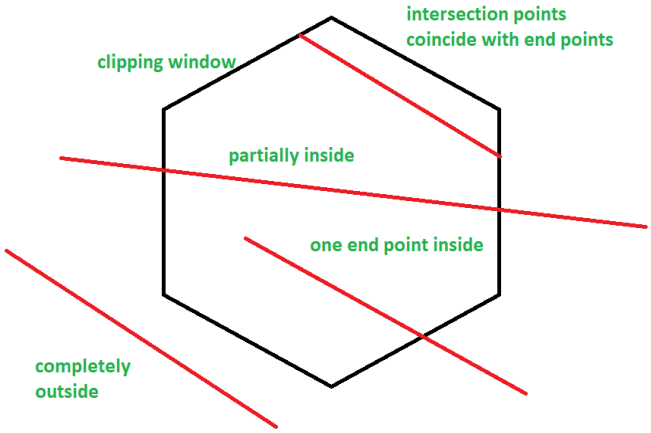
- Normals of every edge is calculated.
- Vector for the clipping line is calculated.
- Dot product between the difference of one vertex per edge and one selected end point of the clipping line and the normal of the edge is calculated (for all edges).
- Dot product between the vector of the clipping line and the normal of edge (for all edges) is calculated.
- The former dot product is divided by the latter dot product and multiplied by -1. This is 't'.
- The values of 't' are classified as entering or exiting (from all edges) by observing their denominators (latter dot product).
- One value of 't' is chosen from each group, and put into the parametric form of a line to calculate the coordinates.
- If the entering 't' value is greater than the exiting 't' value, then the clipping line is rejected.

Cases:

Case 1: The line is partially inside the clipping window:

$$0 < tE < tL < 1$$

where tE is 't' value for entering intersection point
tL is 't' value for exiting intersection point



Case 2: The line has one point inside or both sides inside the window or the intersection points are on the end points of the line:

$$0 \leq tE \leq tL \leq 1$$

Case 3: The line is completely outside the window:

$$tL < tE$$

Pseudocode:

First, calculate the parametric form of the line to be clipped and then follow the algorithm.

- Choose a point called P1 from the two points of the line (P0P1).
- Now for each edge of the polygon, calculate the normal pointing away from the centre of the polygon, namely N1, N2, etc.
- Now for each edge choose PEi (i -> ith edge) (choose any of the vertices of the corresponding edge, eg.: For polygon ABCD, for side AB, PEi can be either point A or point B) and calculate

$$P0 - PEi$$

- Then calculate: P1 - P0
- Then calculate the following dot products for each edge:

$$Ni \cdot (P0 - PEi)$$
$$Ni \cdot (P1 - P0)$$

where i -> ith edge of the convex polygon

- Then calculate the corresponding ‘t’ values for each edge by:

$$t = \frac{Ni \cdot (P0 - PEi)}{-(Ni \cdot (P1 - P0))}$$

- Then club the ‘t’ values for which the Ni . (P1 – P0) came out to be negative and take the minimum of all of them and 1.
- Similarly club all the ‘t’ values for which the Ni . (P1 – P0) came out to be positive and take the maximum of all of the clubbed ‘t’ values and 0.
- Now the two ‘t’ values obtained from this algorithm are plugged into the parametric form of the ‘to be clipped’ line and the resulting two points obtained are the clipped points.

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:

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

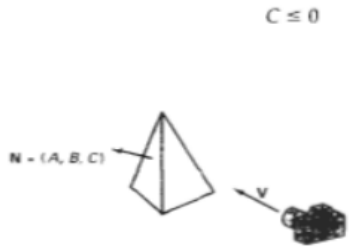
1. Back Face Detection Method:

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



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 who's 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:

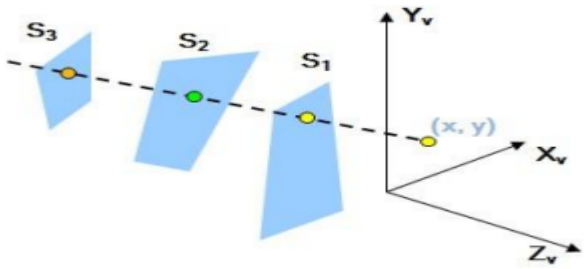
$$C \geq 0$$

2. Depth Buffer Method (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 buffers are required:
   1) Depth Buffer (store's depth value for each pixel)
   2) 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

4. DEPTH SORTING ALGORITHM (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 processes 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

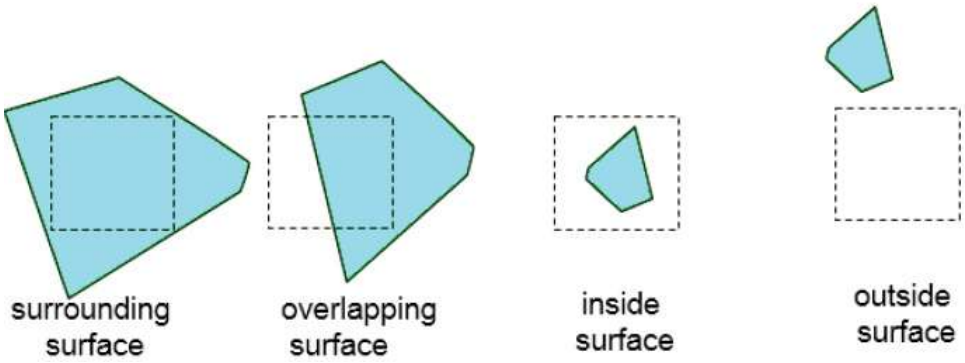
- Tests for each surface that overlaps with S
  - The bounding rectangle in the xy plane for the two surfaces do not overlap
  - Surface S is completely behind the overlapping surface(S') relative to the viewing position
  - The overlapping surface(S') is completely in front of S relative to the viewing position
  - The projections of the two surfaces onto the view plane do not overlap
- 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
- 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

Area-Subdivision Method:

The area-subdivision method takes advantage by locating those view areas that represent part of a single surface. Divide the total viewing area into smaller and smaller rectangles until each small area is the projection of part of a single visible surface or no surface at all.

Continue this process until the subdivisions are easily analyzed as belonging to a single surface or until they are reduced to the size of a single pixel. An easy way to do this is to successively divide the area into four equal parts at each step. There are four possible relationships that a surface can have with a specified area boundary.

- **Surrounding surface** − One that completely encloses the area.
- **Overlapping surface** − One that is partly inside and partly outside the area.
- **Inside surface** − One that is completely inside the area.
- **Outside surface** − One that is completely outside the area.



The tests for determining surface visibility within an area can be stated in terms of these four classifications. No further subdivisions of a specified area are needed if one of the following conditions is true −

- All surfaces are outside surfaces with respect to the area.
- Only one inside, overlapping or surrounding surface is in the area.
- A surrounding surface obscures all other surfaces within the area boundaries.

Back Face Removal Algorithm:

It is used to plot only surfaces which will face the camera. The objects on the back side are not visible. This method will remove 50% of polygons from the scene if the parallel projection is used. If the perspective projection is used then more than 50% of the invisible area will be removed. The object is nearer to the center of projection, number of polygons from the back will be removed.

It applies to individual objects. It does not consider the interaction between various objects. Many polygons are obscured by front faces, although they are closer to the viewer, so for removing such faces back face removal algorithm is used.

When the projection is taken, any projector ray from the center of projection through viewing screen to object pieces object at two points, one is visible front surfaces, and another is not visible back surface.

This algorithm acts a preprocessing step for another algorithm. The back face algorithm can be represented geometrically. Each polygon has several vertices. All vertices are numbered in clockwise. The normal M_1 is generated a cross product of any two successive edge vectors. M_1represent vector perpendicular to face and point outward from polyhedron surface

$N_1 = (v_2-v_1)\ (v_3-v_2)$

If $N_1.P \geq 0$ visible

$N_1.P < 0$ invisible

Advantage:

1. It is a simple and straight forward method.
2. It reduces the size of databases, because no need of store all surfaces in the database, only the visible surface is stored.

Algorithm:

Repeat for all polygons in the scene.

Do numbering of all polygons in clockwise direction i.e. v_1 v_2 v_3.....v_z

Calculate normal vector i.e. N_1

$_{N1}=(v_2-v_1)*(v_3-v_2)$

Consider projector P, it is projection from any vertex Calculate dot product

Dot=N.P

Test and plot whether the surface is visible or not. If Dot ≥ 0 then surface is visible else

Not visible